MODELING OF THE AGING VISCOELASTIC PROPERTIES OF CEMENT PASTE USING COMPUTATIONAL METHODS

A Thesis

by

XIAODAN LI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2012

Major Subject: Civil Engineering

Modeling of the Aging Viscoelastic Properties of Cement Paste Using Computational Methods

MODELING OF THE AGING VISCOELASTIC PROPERTIES OF CEMENT PASTE USING COMPUTATIONAL METHODS

A Thesis

by

XIAODAN LI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Zachary C. Grasley
Committee Members,	Robert L. Lytton
	Anastasia Muliana
Head of Department,	John M. Niedzwecki

May 2012

Major Subject: Civil Engineering

ABSTRACT

Modeling of the Aging Viscoelastic Properties of Cement Paste Using Computational Methods. (May 2012)

Xiaodan Li, Be.N., Hong Kong University of Science and Technology

Chair of Advisory Committee: Dr. Zachary Grasley

Modeling of the time-dependent behavior of cement paste has always been a difficulty. In the past, viscoelastic behavior of cementitious materials has been primarily attributed to the viscoelastic properties of C-S-H components. Recent experimental results show that C-S-H may not exhibit as much creep and relaxation as previously thought. This requires new consideration of different mechanisms leading to the viscoelastic behavior of cement paste. Thus the objective of this thesis is to build a computational model using finite element method to predict the viscoelastic behavior of cement paste, and using this model, virtual tests can be carried out to improve understanding of the mechanisms of viscoelastic behavior.

The primary finding from this thesis is that the apparent viscoelastic behavior due to dissolution of load bearing phases is substantial. The dissolution process occurring during the hydration reaction can change the stress distribution inside cementitious materials, resulting in an apparent viscoelastic behavior of the whole cementitious materials. This finding requires new consideration of mechanisms of time-

dependent behavior of cementitious materials regarding the dissolution process of cement paste.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who helped me in completing this report. First I must give my special thanks to my supervisor Dr. Grasley for his continual encouragement, technical and personal guidance. I also want to thank my committee members, Dr. Lytton and Dr. Muliana, for their guidance and support throughout this research. My friend, Christopher A. Jones and Ardavan Yazdanbakhsh gave me a lot of help when I came across difficulties. Additionally, I would like to thank Dr. E.J. Garboczi and Dr. Jeffrey Bullard for their technical support on the research. Without their support, there is no way I can finish this work. I really appreciate all your guidance, help and patience.

Thanks to the Zachry Department of Civil Engineering of Texas A&M University for giving me the opportunity to start working on this project in the first place, providing me necessary environment to carry out this project and allowing me to use all resources.

Finally, thanks to my mother and my father for their encouragement and support. I love you.

NOMENCLATURE

C-S-H	calcium silicate hydrate
E	elastic Young's modulus
$E(t)$	apparent viscoelastic Young's modulus
K	elastic Bulk modulus
$K(t)$	apparent viscoelastic Bulk modulus
G	elastic Shear modulus
$G(t)$	apparent viscoelastic Shear modulus
υ	elastic Poisson's ratio
$\upsilon(t)$	apparent viscoelastic Poisson's ratio
w/c	water to cement mass ratio
t	time
S_{ijkl}	viscoelastic compliance tensor
σ	stress
ε	strain
C_{ijkl}	elastic moduli tensor
A	Hessian matrix
u	displacement
gb	energy gradient
ρ	mass density
Ω	domain of problem

$v(t)$	aging function

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION

1.1 Motivation

The mechanical properties of cement-based materials, which play a key role in civil infrastructure, have been studied for a long time. Studies have been carried out in developing numerical computational methods to predict their linear elastic properties [1]. However, in reality, cementitious materials are viscoelastic, and to accurately predict stress and strain in cementitious materials, proper understanding of the viscoelastic behavior is necessary. Portland cement concrete by far is the most used construction material worldwide because of its many advantages [2]; thus, understanding the mechanical constitutive behavior of cementitious materials is important for predicting stress and strain in infrastructure.

There are currently several theories that attempt to explain the viscoelastic behavior of cement-based materials, but none of them are capable of describing the viscoelastic behavior adequately. One reason for this is that cement-based materials are composite materials with random matrix arrangement, which are difficult to model; another reason is that the internal chemical components of cement-based materials are changing during the hydration process [3], including dissolution of existent components and forming of new components. The forming of new components in cementitious

---

This thesis follows the style of Cement and Concrete Composites.

materials is also referred to as the aging process. Although there are many other kinds of aging occurring in other viscoelastic materials, in this thesis, the aging process is specified only as the hydration process inside cementitious materials, in which the materials gradually stiffen and their relaxation rates slow down as it "cures" [4].

Viscoelastic behavior of cementitious materials has been primarily attributed to the viscoelastic properties of C-S-H components inside cementitious materials [5]. However, in recent experiments, the results indicate that C-S-H may not exhibit as much creep and relaxation as previously thought [6]. This requires new consideration of different mechanisms leading to the viscoelastic behavior of cement paste. By modeling cement paste's aging viscoelastic behavior, virtual computational experiments can be carried out, and through analyzing the modeling results, the mechanisms causing the viscoelastic behavior of cement paste may be elucidated.

To effectively model cement paste's aging viscoelastic properties, an analytical constitutive model of cement paste based on microstructure can be built, and then the inherent, aging viscoelastic behavior of the micrometric cement paste phases can be integrated to predict the macroscale viscoelastic responses of cement paste.

Through the modeling of aging viscoelastic properties of cement paste, a fundamental understanding of the time dependent behavior of concrete can be reached. In the future, the computational model developed in this research may be used in predicting the behavior of concrete by engineers and to help improve the design of damage-resistant structures. It may also assist in overcoming the current limitations of concrete and developing new advanced materials.

1.2 Scope and Problem Statement

Current limitations with predicting the aging viscoelastic properties of cement paste are that:

- There is no current ability to predict viscoelastic constitutive properties based on cement characteristics;
- There are models linking observed viscoelastic behavior with evolving microstructure linked to hydration process.

The goal of this master's thesis is to:

- Develop a qualified aging viscoelastic composite model for predicting aging viscoelastic properties of cement paste;
- Better understand the mechanisms of viscoelastic behavior through running virtual experiments using the newly developed computational model.

To develop the computational aging viscoelastic composite model, the programming language C++was used with the computational compiling software Microsoft Visual Studio. Since this aging viscoelastic model was a modification of the linear elastic composite materials model developed by Garboczi et al. in the programming language Fortran [1], at the beginning of this project, the programming language Fortran and its compiling software Fortran Powerstation was mainly used.

1.3 Tasks

This thesis research project can be divided into several phases:

- Review Literature;
- Transfer the existent linear elastic model from Fortran to C++;
- Convert the linear elastic model to a viscoelastic model;
- Further develop the model to simulate the dissolution-formation process of cement-based materials;
- Check the accuracy of the finalized model;
- Run virtual experiments and analysis;
- Draw conclusions from virtual experiments regarding mechanisms of viscoelastic behavior of cementitious materials.

1.4 Disposition

This report consists of five chapters. Chapter 2 reviews the previously hypothesized mechanisms for the viscoelastic behavior of cement paste, along with an existent aging viscoelastic model which has some limitations in predicting the behavior of cement paste. Overview of the three dimensional linear elastic model ELAS3D is also included in this chapter. Chapter 3 develops the dissolution-formation viscoelastic model and describes the model in flow charts. Chapter 4 presents the results from the model and includes some discussions and comparisons related to the results. Chapter 5 summarizes the whole report and suggests future work. References are listed at the end

of these chapters. The appendix contains the program code for the dissolution-formation viscoelastic model.

CHAPTER II

LITERATURE REVIEW

In this chapter, hypothesized mechanisms of viscoelastic behavior of cement paste, together with current models are reviewed. In addition, the finite element method and a computational hydration model, which are the basis for developing a qualified aging viscoelastic model, are also reviewed.

2.1 Viscoelastic Properties of Aging Cement Paste

2.1.1 General

There are two different aspects leading to the comprehensive viscoelastic behavior of cement paste, as shown in Figure 1:

First, cement paste's viscoelastic properties are often subdivided into two types: basic components and drying components, which are also referred as 'basic creep' and 'drying creep'. Basic creep is creep in the absence of drying, while drying creep is the additional creep that occurs during drying.

Another critical aspect of viscoelastic behavior of cement paste is the hydration process. During the hydration process, there are changes in internal structure leading to changes in stress and strain distribution. The hydration process can also be divided into two different parts: dissolution and formation. Because of dissolution, the inherent stress will redistribute after dissolution of loaded unhydrated phases and the responses of new

formed components critically depend on load histories relative to the time they are "formed".

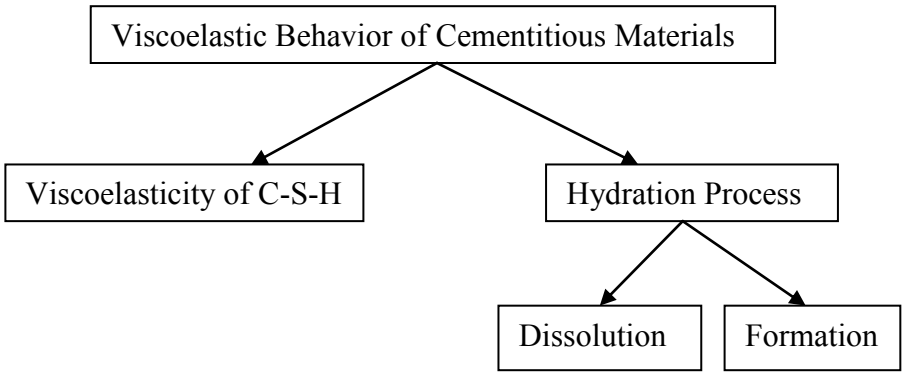

Figure 1 Mechanisms of viscoelastic behavior of cementitious materials.

2.1.2 Viscoelastic Properties of C-S-H

Creep is the tendency of a solid material to deform over some time under the influence of constant stresses. Although all real materials undergo some time-dependent deformation under load, cement paste exhibits larger time-dependent deformation than some other materials, such as ceramics and metals. This is because the fundamental origins for creep of cement paste are different from other materials. In the past, the origins of creep are primarily attributed to the response of C-S-H to stress. Figure 2 shows the typical creep responses of cement-based materials.

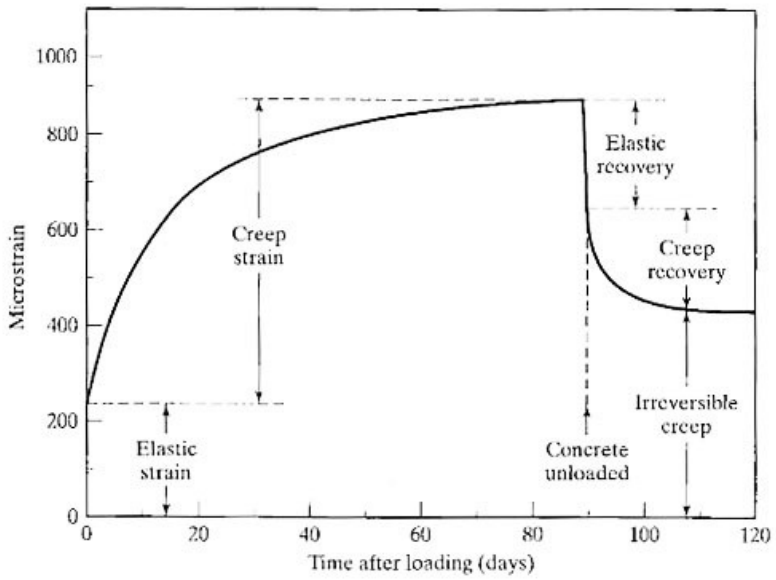

Figure 2 Typical creep curve for plain concrete [5].

There are several proposed mechanisms of viscoelastic behavior of cement paste found in the literature:

- Seepage of physically bound water (within the layers of C-S-H) into capillary water as a result of external load, which is also known as the seepage theory (by Powers) [7, 8];
- Crystallization of C-S-H forming new interlayer space (introduced by Feldman and Sereda) [7, 9];
- Sliding of C-S-H globules or layers under localized nanoscale shear stresses (the viscous shear theory) [7, 10];

2.1.3 Hydration Process

Another critical aspect of viscoelastic behavior of cement paste is the hydration process. Hydration occurs independently of the application of external forces. It is a result of the chemical and physical processes that take place between cement and water. The hydration process causes an inherent component change, and thus the responses of cementitious materials to loads critically depend on load histories relative to the time the new components are "formed" [4].

The main hydration reaction is the reaction between tricalcium silicate and water and between dicalcium silicate and water, and the principal hydration product is C-S-H together with calcium hydroxide. As the hydration process continues, more and more C-S-H is produced and less and less of the silicate phases are left.

Figure 3 illustrates the inherent microstructure change during the hydration process.

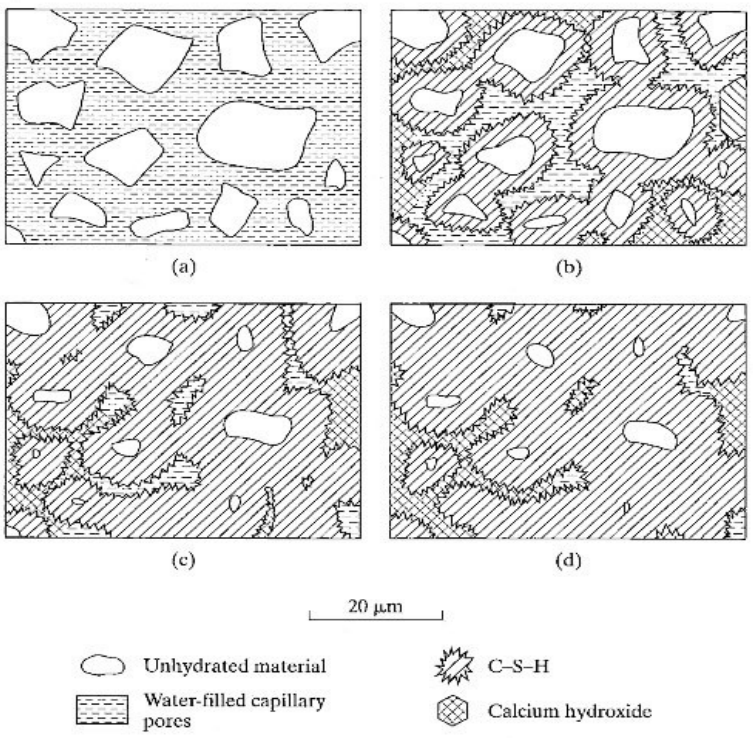

Figure 3 Schematic outline of microstructure development in Portland cement pastes: (a) initial mix; (b) 7 days; (c) 28 days; (d) 90 days. (Calcium sulfoaluminates are included as part of C-S-H for simplification) [5].

As previously shown in Figure 1, the hydration process can be divided into two parts: (a) dissolution of loaded unhydrated phases and (b) forming of new components.

Dissolution of load bearing phases has been proposed as one potential mechanism for the time-dependent deformation of cementitious materials [11]. Mechanisms regarding the aging process have also been considered. Bazant et al. suggested that the microprestress-solidification process appears to be another creep mechanism [7, 12]. The microprestress is generated by disjoining pressure of the hindered adsorbed water in the micropores and by large localized volume changes

caused by hydration. Irreversible creep can be closely related to the aging process due to silicate polymerization induced by drying [7, 13].

2.2 Current Viscoelastic Constitutive Models

Cement paste is known to exhibit time-dependent deformation under different stress histories. A significant amount of research has been carried out to develop valid mathematical models to fit the viscoelastic behavior of viscoelastic materials, and several simple constitutive models have been successfully utilized, such as the Maxwell model, the Kelvin-Voigt model, and the Standard Linear Solid Model [4].

All these models can show accurate simulations for a linear viscoelastic material under different forms of stresses, but the effect of hydration reaction on creep of concrete is not considered in these models. A popular theory which combines linear viscoelasticity and the formation effect is the solidification theory [14, 15], and it will be introduced in detail later in this chapter.

2.2.1 General Constitutive Theory

The most general form of the constitutive equation for a linear viscoelastic material is

$$\varepsilon_{ij}(t) = \int_{o}^{t} S_{ijkl}\,(t - t')\frac{\partial \sigma_{kl}(t')}{\partial t'}dt' + \varepsilon^{f}(t)\delta_{ij}, \qquad (2.1)$$

where S_{ijkl} is the viscoelastic compliance tensor, $\sigma_{kl}(t)$ is the stress function, $\varepsilon^f(t)$ is the free strain, $\varepsilon_{ij}(t)$ is the overall time-dependent strain, and δ_{ij} is Kronecker's delta. In this equation, $S_{ijkl}(t)$ is a function of single variable$(t - t')$.

For aging linear viscoelastic materials, the most general form is

$$\varepsilon_{ij}(t) = \int_o^t S_{ijkl}(t, t') \frac{\partial \sigma_{kl}(t')}{\partial t'} dt' + \varepsilon^f(t)\delta_{ij}, \tag{2.2}$$

where $S_{ijkl}(t)$ is a function of both *t* and t'. Some well-known constitutive models coupling the viscoelasticity and aging effect are essentially different forms of eq. (2.2), such as the time-shift theory and the solidification theory [16].

2.2.2 Solidification Theory

The solidification theory is a theory accounting for the formation effect on creep of concrete. This theory implies that hydration products are non-aging viscoelastic materials and aging presents in the bulk scale due to the increase in load bearing materials as a result of solidification and deposition of hydration products [17]. As hydration progresses, each newly-formed layer of the hydration products solidifies in a stress-free state and these layers are only subject to loads after they form. This theory is shown conceptually in Figure 4.

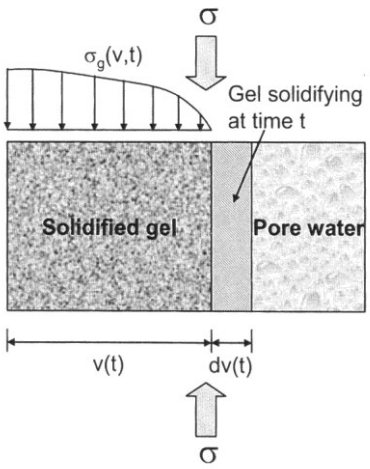

Figure 4 Conceptual diagram of solidifying material subjected to applied stress, σ. The volume of solidified gel is dependent on the aging function $v(t)$ (which is representative of the volume of load-bearing products that have formed) and $\sigma_g(v,t)$ is the stress distribution carried by solidified phases depending on the aging function and time [15, 16].

According to the solidification theory, the constitutive equation for a uniaxial load is

$$\frac{\partial \varepsilon_{11}(t)}{\partial t} = \frac{J_0}{v(t)}\frac{\partial \sigma_{11}(t)}{\partial t} + \frac{1}{v(t)}\int_0^t \frac{\partial J_g(t-t')}{\partial t'}\frac{\partial \sigma_{11}(t')}{\partial t'}dt', \qquad (2.3)$$

Where J_0 *i*s the instantaneous elastic uniaxial compliance , $\sigma_{11}(t)$ is the applied uniaxial stress at time *t*, $v(t)$ is the aging function and $J_g(t-t')$ is the non-aging viscoelastic uniaxial compliance [14].

Another form of the constitutive equation for a solidifying, linear viscoelastic material is expressed as:

$$\sigma_{11}(t) = \int_{0^-}^{t} \nu(t^{'}) \frac{\partial \varepsilon_{11}(t^{'})}{\partial t^{'}} E_g(t - t^{'}) dt', \tag{2.4}$$

where $\nu(t)$ is the aging function, $E_g(t - t^{'})$ *i*s the non-aging viscoelastic uniaxial moduli of the solidifying phases [18].

The solidification theory has a significant limitation when applied to concrete. For cementitious materials, formation of new load bearing phases (such as C-S-H) requires dissolution of existing loading bearing phases (such as tricalcium silicate). The solidification theory predicts the formation process well but it does not consider the dissolution of existing load bearing phases, as a result, the stress carried by new load bearing phases is gradually decreasing according to solidification theory. However, when accounting for dissolution, load is transferred gradually from unhydrated phases to newly formed phases and the stresses carried by these new phases are *increasing*. A detailed comparison between the solidification theory and the dissolution-formation viscoelastic model developed in this thesis will be shown in Chapter 4.

2.2.3 Mathematical and Physical Representation of Modulus and Compliance

As mentioned at the beginning of this chapter, there are many mathematical models that can simulate the moduli of linear viscoelastic materials. Two of the most popular forms are the Kevin chain and the Maxwell Chain, which are shown in Figure 5.

The formula for expressing the Kevin chain is

$$J(t) = \sum_{i=1}^{I} \frac{1}{R_i} \left(1 - e^{-\frac{t}{\tau_i}}\right), \tag{2.5}$$

where $J(t)$ is the viscoelastic compliance of the whole system, $\tau_i = \eta_i/R_i$ are called the retardation times, R_i is the elastic Young's modulus of spring *i* in the model and η_i is the viscosity coefficient of damper *i* [4].

The formula for expressing the Maxwell chain is

$$E(t) = \sum_{i=1}^{I} R_i(e^{-\frac{t}{\tau_i}}), \tag{2.6}$$

where $E(t)$ is the viscoelastic Young's modulus of the whole system, $\tau_i = \eta_i/R_i$ are the relaxation times, R_i is the elastic Young's modulus of spring *i* in the model and η_i is the viscosity coefficient of damper *i* [4].

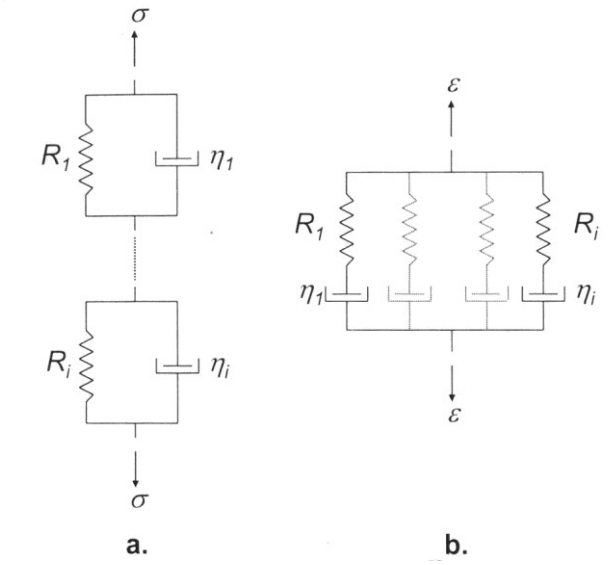

Figure 5 The Kelvin Chain (a.) and the Maxwell Chain (b.) [16]. R_i are springs' Young's modulus and η_i are dampers' viscosity coefficients.

Since the Kelvin Chain is ideal for modeling creep and the Maxwell Chain is ideal for modeling stress relaxation, for the dissolution-formation viscoelastic model in

this thesis, which is applied under strain controlled conditions, the Maxwell Chain is used.

2.3 Linear Elastic Model Using Finite Element Method

2.3.1 Finite Element Method

The finite element method is a general technique for constructing approximate solutions to boundary-value problems [19].

The finite element method is the solution technique used for both elastic models and viscoelastic models presented in this thesis. It involves dividing the domain of a solution into a finite number of simple subdomains, and constructing an approximation of solution over the collection of finite elements.

Thus, for a macro-scaled cement paste microstructure, the 3D composite is divided into a finite number of micro-scaled voxels, and each voxel has its own mechanical properties due to the complexity of cement-based materials. Numerical analysis is carried out on these finite number of voxels. When boundary conditions are applied, for example, a fixed load applied on the boundary of the microstructure, voxels will deform according to the load. Using the finite element method, the displacement of each voxel can be calculated, and thus the total composite mechanical response of the whole microstructure.

2.3.2 Linear Elastic Composite Model

There is currently a linear elastic composite model developed by Garboczi et al. [1, 20, 21] for predicting the linear elastic behavior of cement paste using the finite element method. This three dimensional elastic model is also referred to as the ELAS3D model.

The hydration model CEMHYD3D is used to generate the microstructure for this ELAS3D model at the micrometer length scale. The output of the CEMHYD3D model is a 3-D digital microstructure where each voxel is labeled with a single phase at the dimension of 1 μm^3. ELAS3D treats each cubic voxel as a tri-linear finite element and the elastic equations are solved using a relaxation algorithm [22].

In this finite element model, elastic moduli for each voxel used are shown in Table 1.

Table 1 Elastic moduli of individual cement and cement paste phases [1].

Elastic moduli of individual cement and cement paste phases, taken from several sources in the literature

Cement chemistry notation	Mineral name	K (GPa)	G (GPa)	E (GPa)	υ
H	Water	2.2	0.0	–	–
C_3S	Tricalcium silicate	105.2	44.8	117.6	0.314
C_2S	Dicalcium silicate	Same as C_3S			
C_3A	Tricalcium aluminate	Same as C_3S			
C_4AF	Tetracalcium aluminoferrite	Same as C_3S			
$C\overline{S}\cdot H_2 (gypsum)$	Dihydrate	42.5	15.7	45.7	0.33
$C\overline{S}\cdot H_{1/2}$	Hemihydrate	52.4	24.2	62.9	0.30
$C\overline{S}$	Anhydrite	54.9	29.3	80.0	0.275
$K\overline{S}$	Potassium sulfate (arcanite)	31.9	17.4	44.2	0.269
$N\overline{S}$	Sodium sulfate (thenardite)	43.4	22.3	57.1	0.281
SiO_2	Silica fume	36.5	31.2	72.8	0.167
CH	Portlandite	40.0	16.0	42.3	0.324
$C_{1.7}SH_4$	C–S–H	14.9	9.0	22.4	0.25
$CaCO_3$	Limestone	69.8	30.4	79.6	0.31
C_3AH_6	Hydrogarnet	Same as C–S–H			
$C_6A\overline{S}_3H_{32}$	Ettringite	Same as C–S–H			
$C_4A\overline{S}H12 (Afm)$	Monosulfate	Same as CH			
FH_3	Iron hydroxide	Same as C–S–H			
$CaCl_2$	Calcium chloride	Same as CH			
$C_3A(CaCl_2)H_{10}$	Friedel salt	Same as ettringite			
C_2ASH_8	Stratlingite	Same as C–S–H			
$C_3A(CaCO_3)H_{11}$ (Afmc)	Monocarbonate	Same as Afm			

For an isotropic elastic material, with any two of the set: the Young's modulus, E, the Poisson's ratio, ν, the Bulk modulus, K, and the Shear modulus, G, the remaining two elastic moduli can be calculated through

$$K = \frac{E}{3(1-2\nu)} \qquad G = \frac{E}{2(1+\nu)}. \tag{2.7}$$

These equations can be used to solve elastic moduli with different known combinations of these parameters [23, 24].

The ELAS3D model is a strain controlled model with periodic boundary conditions. Strain controlled means that the bulk strain applied on the whole system is kept constant and periodic boundary conditions mean that if there is a neighbor outside the digital image, the model is periodically continued on the opposite side of the system.

2.3.2.1 Theory

The theory on which the finite element program ELAS3D is based is simple. For a given microstructure under applied boundary conditions, such as a controlled strain, the final displacement distribution is calculated such that the total energy stored in the elastic microstructure is minimized. Another way of expressing the essential idea of this finite element program is that the gradient of energy with respect to elastic displacement is zero:

$$Grad(En) = 0, \tag{2.8}$$

where En is the elastic energy stored.

In this finite element program, the sum of the squares of the gradient vector of all elements is forced to be less than a prescribed small value, so that the condition expressed in eq. (2.8) is approximately satisfied.

For an elastic microstructure, the total elastic energy is given by

$$En = \frac{1}{2}\int_0^1 \int_0^1 \int_0^1 \varepsilon_{pq}\, C_{pqrs}\, \varepsilon_{rs}\, dx\, dy\, dz, \tag{2.9}$$

where ε_{pq} is the strain tensor and C_{pqrs} is the elastic moduli tensor, *p, q, r, s* = 1, 2, or 3, and the integral is over the volume of a single unit voxel.

By expressing the strain tensor in terms of displacement components, eq. (2.9) can be rewritten as

$$En = \frac{1}{2} u_{rp}^T D_{rpsq}\, u_{sq}, \tag{2.10}$$

where D_{rpsq} is the stiffness matrix and u_{rp} is the p'th component of displacement at r'th node.

Since periodic boundary conditions are applied, for a voxel at the boundary, displacement of its nodes: $u_{rp} = U_{rp} + \delta_{rp}$, where U_{rp} is the displacement vector determined by surrounding voxels and δ_{rp} is the correction vector determined by boundary conditions.

Inserting these boundary conditions, the expression for energy becomes

$$En = \frac{1}{2}\left[u_{rp}^T D_{rp,sq}\, u_{sq} + 2\delta_{rp} D_{rp,sq}\, u_{sq} + \delta_{rp} D_{rp,sq}\, \delta_{sq}\right], \tag{2.11}$$

which can be simplified as

$$En = \frac{1}{2}uAu + bu + C, \tag{2.12}$$

where *A* is the Hessian matrix composed of the stiffness matrices, *b* is a constant vector and *C* is a constant that is determined by the applied strain and periodic boundary conditions, and *u* is a vector of all the displacements.

The only contributions to *b* and *C* come from voxels having nodes at the unit cell boundaries and having a non-zero stiffness matrix. In other words, *b* and *C* are determined by boundary conditions.

When integrating the energy inside each voxel, since there is no term to be integrated that is higher order than quadratic, Simpson's rule is used for solving the exact solution of the quadratic functions.

The solution method used is the conjugate gradient relaxation algorithm:

$$\frac{\partial En}{\partial u} = gb\ (gradient) = \ Au + b. \tag{2.13}$$

By changing the values of *u*, the final results of *u* can be found when $gg = gb * gb$ is close enough to zero or when the program has been run enough times. When *gg* is close enough to zero, it means that the total energy of the system approaches a minimum.

2.3.2.2 Flow Chart

Figure 6 shows the procedural flow chart for the elastic finite element program:

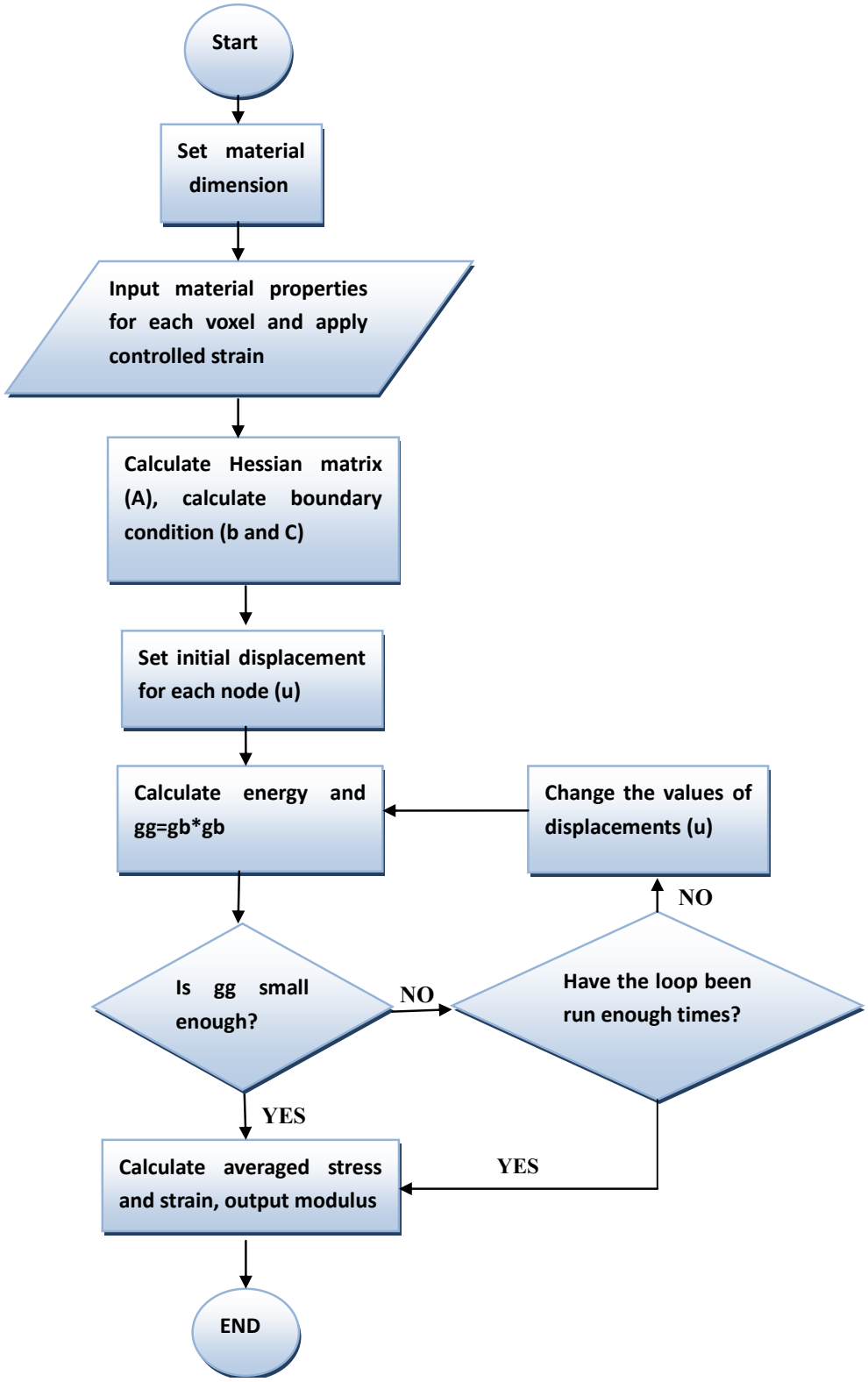

Figure 6 Complete flow chart for finite element linear elastic composite model.

The flow chart can also be simplified as Figure 7:

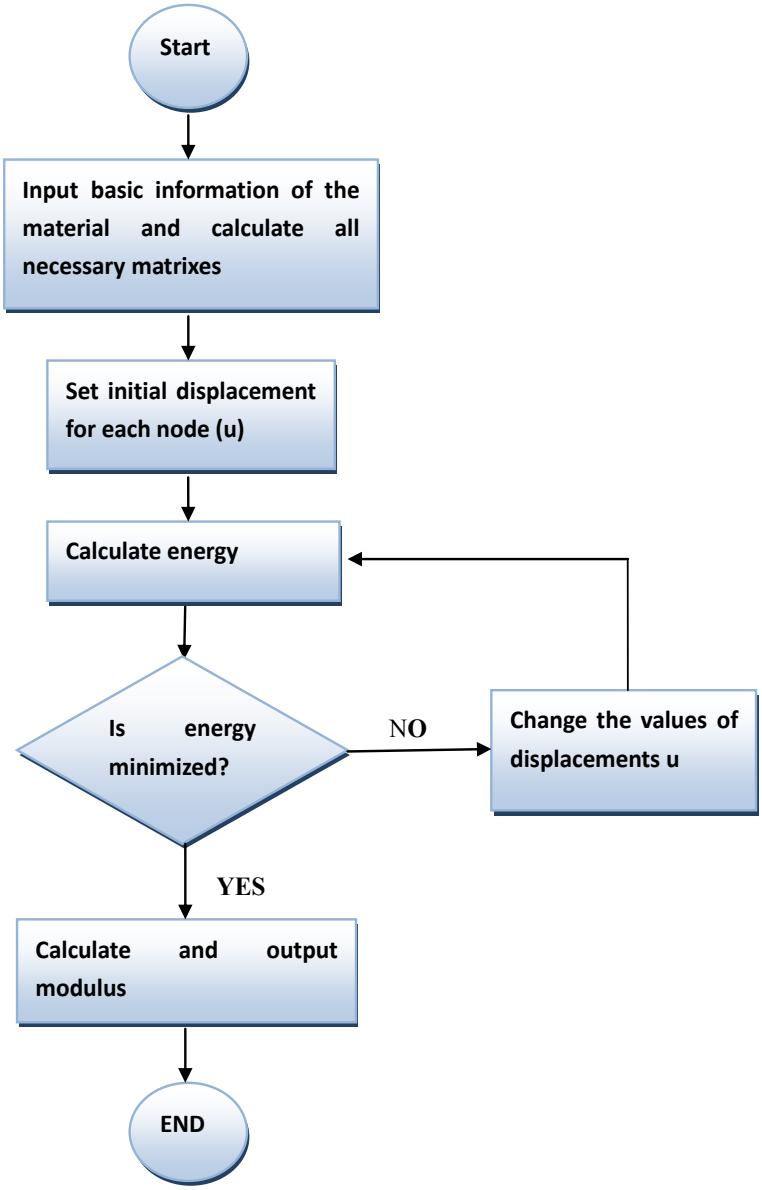

Figure 7 Simplified flow chart for finite element linear elastic composite model.

2.4 Hydration Model

In this thesis, instead of CHMHYD3D model, a complementary numerical model called THAMES [25, 26] is used to simulate the system of cement hydration and microstructure development. It relies heavily on equilibrium thermodynamic calculations to predict the assemblage of phases and then using a digital image representation, it distributes these phases into a 3D microstructure model.

To predict time-dependent phase assemblage in hydrating cement pastes, equilibrium thermodynamic calculations coupled with a kinetic model for the dissolution rates have been used. Combining this approach with a lattic-based 3D microstructure model, microstructure development during the hydration process can be simulated. This is the basis for the THAMES model. Using the finite element method on one 3D microstructure image, as shown in this thesis, effective elastic and viscoelastic moduli can be calculated [25-27].

2.5 Viscolasticity of C-S-H

C-S-H has different mechanical properties on the nanometric scale versus on the micro scale by different test results based on AFM (atomic force microscopy) tests and nanoindentation measurements. [28-30]

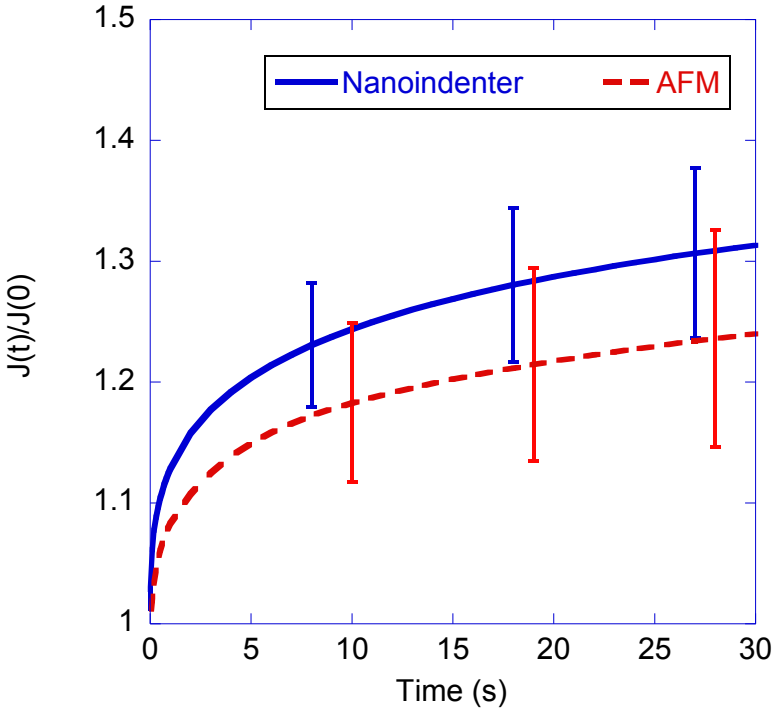

Figure 8 Uniaxial viscoelastic compliance of C-S-H measured using a traditional nanoindenter and an AFM; values are normalized by instantaneous uniaxial compliance and vertical bars indicate one standard deviation [31].

The measured uniaxial viscoelastic compliances of cement paste measured using both nanoindenter and AFM are shown in Figure 8. Measurements with the nanoindenter yield a higher creep rate than measurements performed with an AFM. A possible explanation for the higher creep rate exhibited in the nanoindenter test is that multiple creep processes are present in the nanoindenter test, while only inherent viscoelasticity of C-S-H is present in the AFM test because of its smaller test volume. The nanoindenter is more likely probing multiple phases (both hydrated and unhydrated) simultaneously [32, 33]. Thus, Figure 8 suggests that, since more creep is observed with a larger volume of material interacted, C-S-H viscoelasticity cannot be the sole

mechanism behind bulk (e.g. meter scale) viscoelastic response of cementitious materials.

CHAPTER III

COMPUTER MODELING THEORY

This chapter describes the theory and steps to develop a qualified dissolution-formation viscoelastic model using the finite element method. Model inputs and flow chart of the simulation program are shown in detail in this chapter.

The development of the computational dissolution-formation viscoelastic model contains three main parts: (1) build a linear elastic composite model using C++; (2) modify the linear elastic composite model so that it can simulate the behavior of linear viscoelastic composite materials; (3) further develop the finite element model to predict the dissolution-formation viscoelastic behavior of cement paste. Through the whole program, the elastic moduli used for different phases are shown in Table 1.

3.1 Elastic

The elastic model developed in this research is the same as the linear elastic composite model ELAS3D. The program was transformed from Fortran to C++, because the programming language C++ is more flexible and is capable of more complex calculations.

The elastic program shows the same calculation theory and flow chart as the linear elastic program shown in Chapter 2.

3.2 Viscoelastic

Modification of the elastic program is needed for the program to predict the viscoelastic behavior of composite materials. It is known that viscoelastic response is a long-term history-dependent response. Different load histories lead to different responses. To accurately predict cement paste's viscoelastic properties, the concept of time steps is used in the model. This model divides a continuous time into a finite number of discrete time steps and assumes the modulus of C-S-H to stay constant during each time step. In this way, the viscoelastic problem can be solved at different time steps, and simulation of viscoelastic behavior becomes achievable.

3.1.2 Theory

As mentioned in section 2.3, the main method for finding the inherent displacement and stress fields in the finite element method is to minimize the total energy stored inside the whole microstructure. Since viscoelastic response is not a static response, to calculate the energy of microstructure, the principle of virtual work or virtual velocities, demonstrated in Figure 9, can be used to derive the equation expressing the energy of viscoelastic materials.

3.1.2.1 Virtual Work

In this model, the derived energy equation is assumed to be applied under a quasi-static state with a negligible inertial. Depending on this, the derivation of the energy equation is shown below:

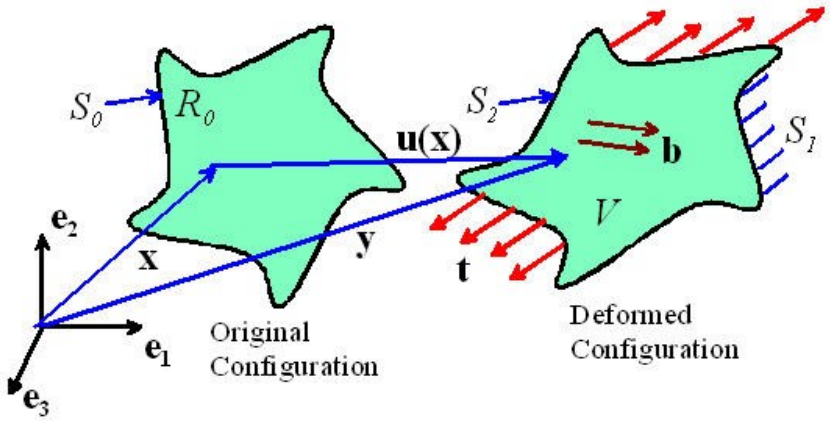

Figure 9 Virtual work demonstration [34].

Consider a solid with mass density ρ, the equilibrium equation for balance of linear momentum is

$$\sigma_{ji,j} + b_i = \rho\ddot{u}_i, \quad i,j = 1,2,3 \tag{3.1}$$

where $\sigma_{ji,j}$ are components of Cauchy Stress [35, 36], ρ is mass density, b_i are body force components and u_i are displacement components.

According to balance of angular momentum, $\sigma_{ji} = \sigma_{ij}$.

For all points in the domain of the problem Ω, stress boundary conditions are given by

$$t_i = \sigma_{ji} n_j = \overline{t_i} \tag{3.2}$$

for all points lying on the boundary S_2.

From the virtual work principle, the total work done by external forces are

$$\int_{\Omega} \delta u_i b_i d\Omega + \int_{S_2} \delta u_i \overline{t_i} d\Omega. \tag{3.3}$$

where, as mentioned above, Ω is the domain of the problem and b_i are body force components.

For the work change inside the material, one component is the change of kinetic energy, the other component is the work done by Cauchy stresses, which is either dissipated as heat or stored as internal energy. This change is

$$\int_{\Omega} \delta u_i \rho \ddot{u}_i d\Omega + \int_{\Omega} \delta \epsilon_{ij} \sigma_{ij} d\Omega, \tag{3.4}$$

where $\delta \epsilon_{ij} = \frac{1}{2}\left(\delta u_{i,j} + \delta u_{j,i}\right)$ for infinitesimal displacement gradients.

These terms in eq. (3.3) should be equal to those in eq. (3.4), thus:

$$\int_{\Omega} \delta u_i b_i d\Omega + \int_{S_2} \delta u_i \overline{t_i} d\Omega - \int_{\Omega} \delta u_i \rho \ddot{u}_i d\Omega - \int_{\Omega} \delta \epsilon_{ij} \sigma_{ij} d\Omega = 0. \tag{3.5}$$

As mentioned at the beginning of this section, for long-term relaxation simulation, the velocity can be approximated as zero, thus eq. (3.5) becomes

$$\int_{\Omega} \delta u_i b_i d\Omega + \int_{S_2} \delta u_i \overline{t_i} d\Omega - \int_{\Omega} \delta \epsilon_{ij} \sigma_{ij} d\Omega = 0. \tag{3.6}$$

The above equation can be rewritten as:

$$\int_{\Omega} \delta u b d\Omega + \frac{1}{2} \int_{\Omega} \delta u \sigma d\Omega = \int_{\Omega} \delta \varepsilon \sigma d\Omega, \tag{3.7}$$

since for small deformation, $\frac{\delta u}{r} = \varepsilon$, where r is the dimension of one voxel, which equals one, thus $\delta u = \varepsilon$ and eq. (3.7) becomes

$$En \approx b\varepsilon + \frac{1}{2}\sigma\varepsilon + Constant. \tag{3.8}$$

Therefore, the energy for a viscoelastic material can be expressed as

$$En = \frac{1}{2}\sigma\varepsilon + b\varepsilon + C. \tag{3.9}$$

3.1.2.2 Computational Theory

In linear elastic problems, $\sigma_{ij} = C_{ijkl}\,\varepsilon_{kl}$, where the elastic moduli tensor C_{ijkl} is constant, and stress and strain are linearly related.

However, in linear viscoelastic problems,

$$\sigma_{ij}(t) = \int_0^t \varepsilon_{kl}(t-s)\dot{C}_{ijkl}(s)ds = \int_0^t C_{ijkl}(t-s)\dot{\varepsilon}_{kl}(s)ds, \tag{3.10}$$

where $C_{ijkl}(t)$ is moduli tensor at time *t* and *i, j, k, l* = 1, 2, or 3. It can be seen that the relation between stress and strain is history dependent. Thus to solve viscoelastic problems, the energy expression becomes complicated and time steps are used. Using the concept of time steps, the continuous eq. (3.10) can be rewritten as

$$\begin{aligned}\sigma_{ij}(t_n) = {} & \varepsilon_{kl}(t_n)C_{ijkl}(t_0) \\ & + \sum_{k=0}^{k=n-1} \varepsilon_{kl}(t_k)[C_{ijkl}(t_n - t_k) - C_{ijkl}(t_n - t_{k+1})],\end{aligned} \tag{3.11}$$

where t_0 is the initial time when load is applied and t_i is the time at *i*th time step .

Since in this program, time steps are set to be equal to each other, eq. (3.11) can be simplified as

$$\begin{aligned}\sigma_{ij}(t_n) = {} & \varepsilon_{kl}(t_n)C_{ijkl}(t_0) \\ & + \sum_{k=0}^{k=n-1} \varepsilon_{kl}(t_k)[C_{ijkl}(t_{n-\mathrm{k}}) - C_{ijkl}(t_{n-k-1})].\end{aligned} \tag{3.12}$$

From eq. (3.12), it can be seen that to calculate the stress for each time step, the material properties and the displacements history should all be stored in the program for calculation, which is a computationally demanding task.

After modifying the equation for calculating stress, the expression for calculating the energy gradient, gb, becomes

$$gb = u(t_n)A(t_0) + \sum_{k=0}^{k=n-1} u(t_k)[A(t_{n-k}) - A(t_{n-k-1})] + b(t_n), \quad (3.13)$$

where $b(t_n)$ is obtained from the boundary conditions depending on the stiffness properties of the materials at time t_n.

In eq. (3.13), since displacements from previous time steps are fixed, *gb* becomes a linear equation with respect to $u(t_n)$, and the equation can be solved in the same way as in elastic case. Eq. (3.13) can be rewritten as

$$gb = u(t_n)A(t_1) + b(t_n), \quad (3.14)$$

where $b(t_n)$ contains all the fixed components in eq. (3.13) at time equals t_n.

After modifying the ELAS3D program by incorporating time steps and modifying the equations of energy, boundary conditions, energy gradient and stress, the program can be used to solve viscoelastic problems.

3.2.2 Flow Chart

The procedural flow chart for the linear viscoelastic program is shown in Figure 10.

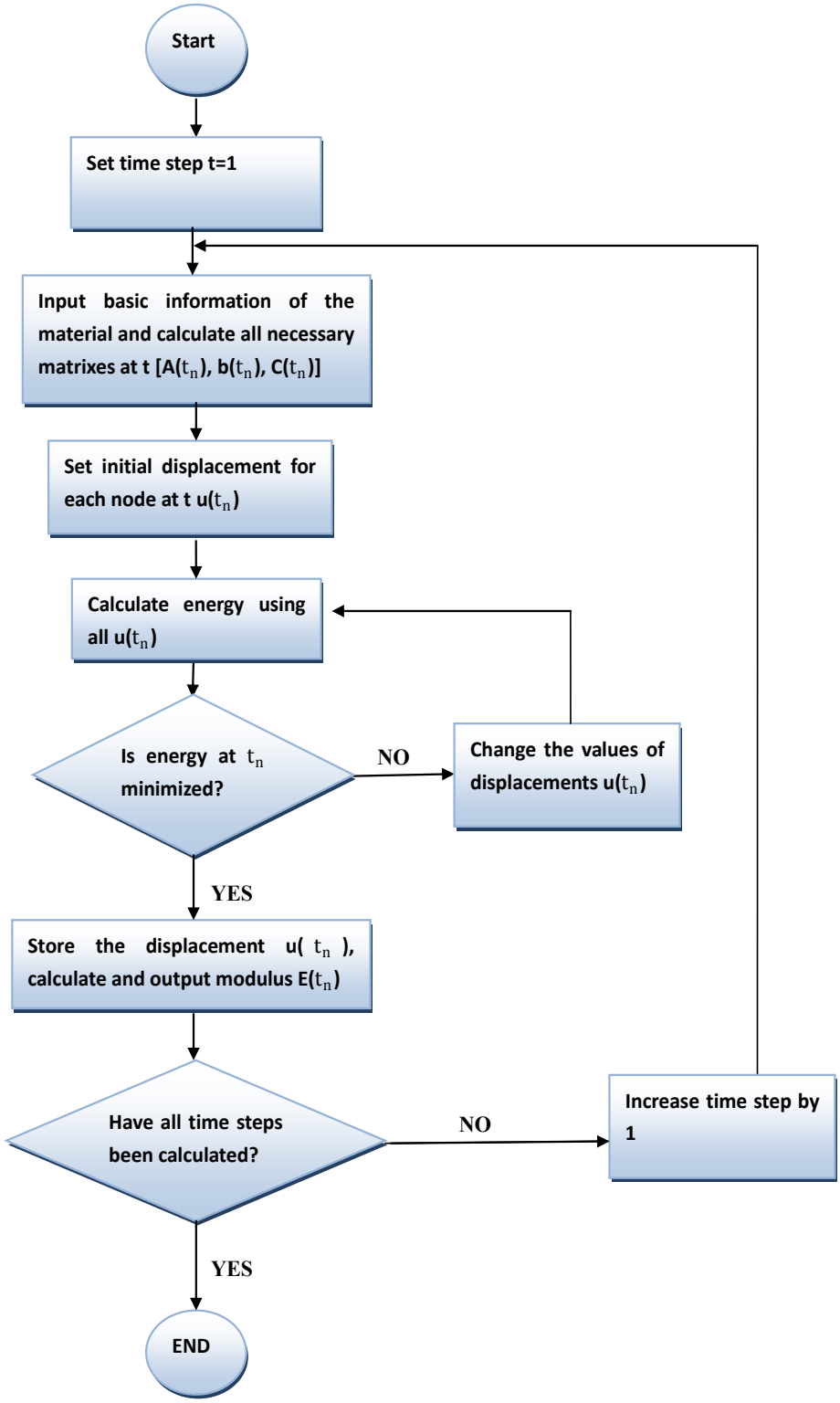

Figure 10 Flow Chart for linear viscoelastic model.

3.3 Dissolution and Formation

3.3.1 Theory

Both hydration and viscoelastic behavior are history dependent behavior, thus time steps should also be used to numerically account for history effects of dissolution and formation processes.

The process of dissolution and formation can be divided into several steps illustrated conceptually in Figure 11:

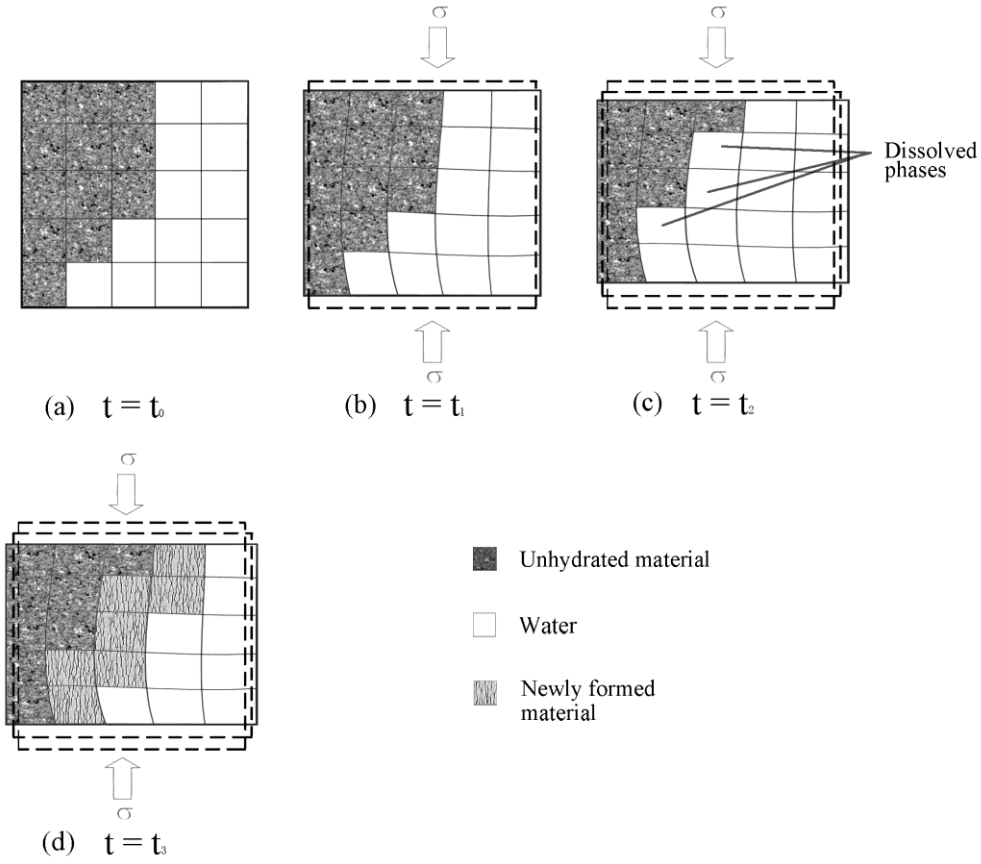

Figure 11 Conceptual diagram of computational model simulating apparent viscoelastic behavior of hydrating cement paste where σ is a constant applied external macroscale stress and t is time elapsed.

In Figure 11, the steps are:

(a) Before load is applied and there are a certain amount of load-bearing unhydrated materials phases;

(b) Load is applied and inherent components deforms under stresses;

(c) After some time, some unhydrated material dissolves, increasing the stress in surrounding phases. Inherent stress and strain redistribute to maintain the boundary condition and the overall deformation of the composite microstructure changes.

(d) New components form in the deformed configuration with free strain and no stresses;

The main challenge in dealing with dissolution-formation problem is that after formation occurs, no historical behavior before formation should be considered and free strain is produced inside microstructure. That is, newly formed phases form in stress-free state that conforms to the preexisting deformed geometry. Thus to correctly predict the behavior of such materials, information regarding free strain and the time at which aging occurs should all be stored in the program.

The energy gradient can still be written as a linear equation

$$gb = u(t_n)A(t_0) - u_f A(t_0) + b(t_n), \tag{3.15}$$

where $b(t_n)$ contains the values calculated from periodic boundary conditions, values calculated from viscoelastic histories, and u_f is the free strain.

Or the energy gradient can be written as

$$gb = u(t_n)A(t_0) + b(t_n), \tag{3.16}$$

where $b(t_n)$ contains all the fixed variables at time equals t_n, including the values calculated from periodic boundary conditions, values calculated from viscoelastic histories, and values calculated from free strains.

Some modifications should also be made to the energy equation and to the constitutive equation. The overall process is similar with the process of linear viscoelastic problems.

3.3.2 Flow Chart

The procedural flow chart for the dissolution-formation viscoelastic program is shown in Figure 12.

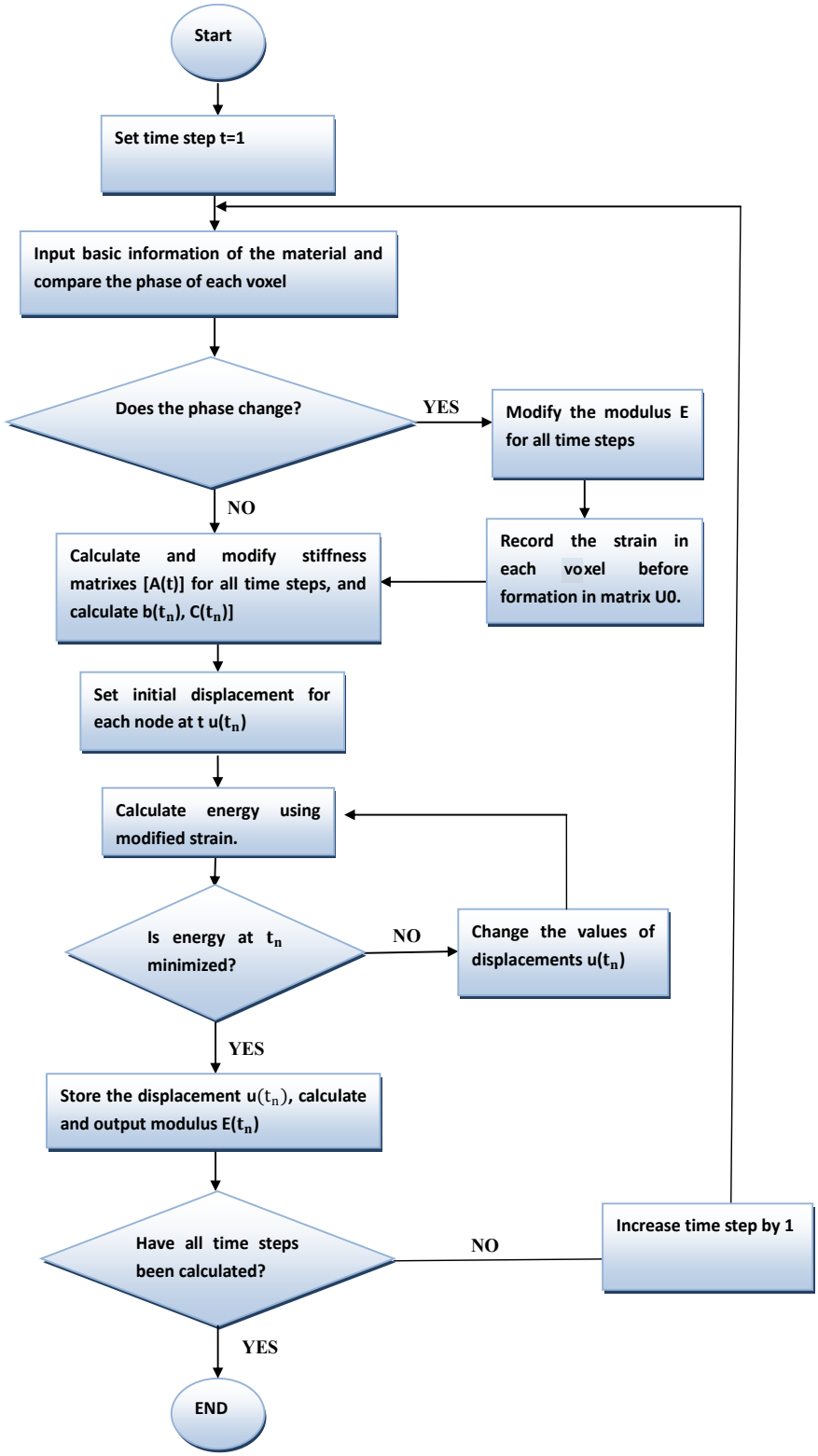

Figure 12 Flow Chart for dissolution-formation viscoelastic model.

CHAPTER IV

RESULTS AND DISCUSSION

In this chapter, the results from the dissolution-formation viscoelastic program are shown in graphs, and discussion about these results will be presented. First, the validation of the program is checked, and then detailed analysis and comparisons of the results are presented to gain a deeper understanding towards the mechanisms of the viscoelastic behavior of cement paste.

4.1 Validation of Model

4.1.1 Elastic

Elastic cases were tested for the dissolution-formation viscoelastic program at the first step, and the results from the program were compared with the results from the ELAS3D model and Abaqus.

First, since the dissolution-formation viscoelastic program in this thesis was a modification based on the NISTIR ELAS3D program, which was developed by Garboczi et al. [1], results from the dissolution-formation viscoelastic program were first compared with the results from the NISTIR ELAS3D program for elastic cases. The purpose of this procedure was to make sure that the translation from FORTRAN to C++ was correct, and there were no errors in the elastic part of the program.

Next, the results from the dissolution-formation viscoelastic program were compared with the results from the program Abaqus. Abaqus is a well-known

commercial finite element program which shows good simulation of mechanical behavior of materials. By checking the results from the program with the results from Abaqus, the accuracy of the program can be further ensured.

Figure 13 shows the comparison results from the three programs: NISTIR ELAS3D, Abaqus and the dissolution-formation viscoelastic program developed in this thesis.

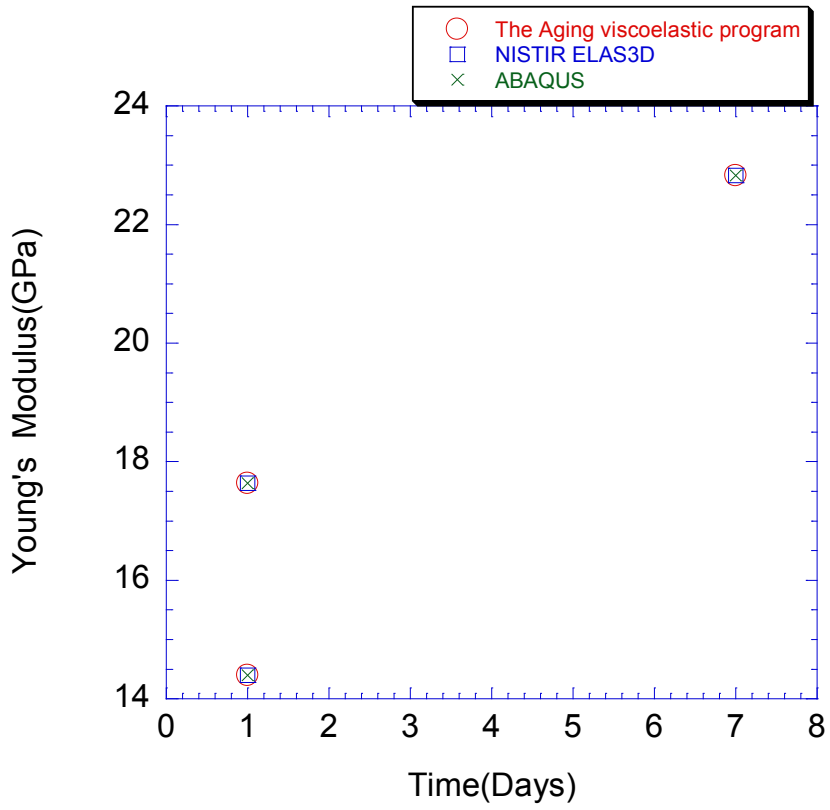

Figure 13 Comparison of elastic results from the dissolution-formation viscoelastic program, the NISTIR ELAS3D program and the Abaqus program for cement paste with w/c of 0.40 and 0.45 at different loading ages.

It can be seen from Figure 13 that the results from the three programs match with each other, and the accuracy of elastic part of the dissolution-formation viscoelastic program can be guaranteed.

4.1.2 Viscoelastic

For the viscoelastic part of the dissolution-formation viscoelastic program, first the results from the program for some special cases were checked. Second, the results from the program were compared with the results from Abaqus on non-aging linear viscoelastic composite materials.

Since the model is a composite materials model, for some simple cases, the results from the program can be compared with analytical results. For example, if all components of the composite carry the same material phases, the final modulus of the whole microstructure should be the same as the modulus of this material. The dissolution-formation viscoelastic program showed satisfying results for these special cases.

Second, the viscoelastic results from this program were also compared with the results from Abaqus for simple random non-aging composites. A random three dimensional non-aging composite matrix with the dimension 20 by 20 by 20 was generated in a *Mathematica* program, and then this composite matrix was analyzed with assigned component properties in Abaqus. The same matrix and component properties were used in the dissolution-formation viscoelastic program and the results from these two programs were compared. Figure 14 shows the comparison results, and there are consistencies between these two programs for the simple non-aging matrix.

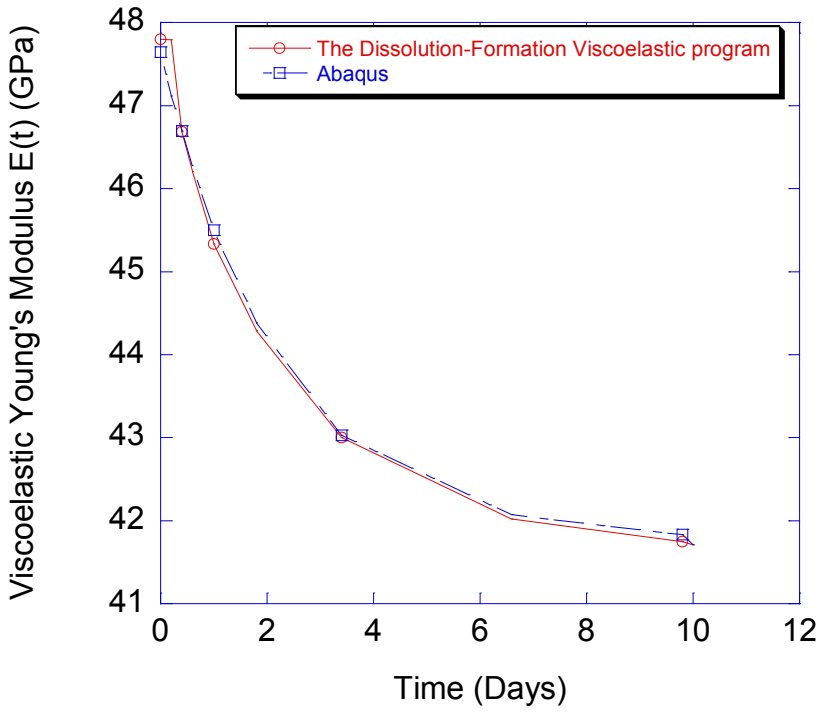

Figure 14 Comparison of viscoelastic results from the dissolution-formation viscoelastic program and the Abaqus program, where phases are either elastic or linear viscoelastic and microstructure is unchanging.

4.1.3 Dissolution-Formation

Currently, there are no valid ways of checking the accuracy of the dissolution-formation part of this viscoelastic model. However, to confirm the accuracy of the dissolution-formation part of the program, some simple special cases were tested.

The main difficulty for predicting the dissolution-formation process is the stress redistribution after the dissolution of existing load bearing phases and the formation of stress-free solidified phases. To prove this dissolution-formation viscoelastic program can give realistic simulation at this part, a simple virtual test was carried out.

Consider an example where there is one cement composite matrix, and the C-S-H phase inside it is assigned a viscoelastic Young's modulus of $11.2 + 11.2e^{-0.2t}$, where t is in days, and a constant Poisson's ratio of 0.25, while all other components of the composite material are assumed to carry the same elastic properties as shown in Table 1. Then a virtual strain history shown in Figure 15 is applied on this composite. During the first 13 days, a constant bulk strain of 0.1 is applied on the composite material, and at the 13th day, there is a sudden removal of the strain and the material is forced to return back to its original dimension of zero bulk strain.

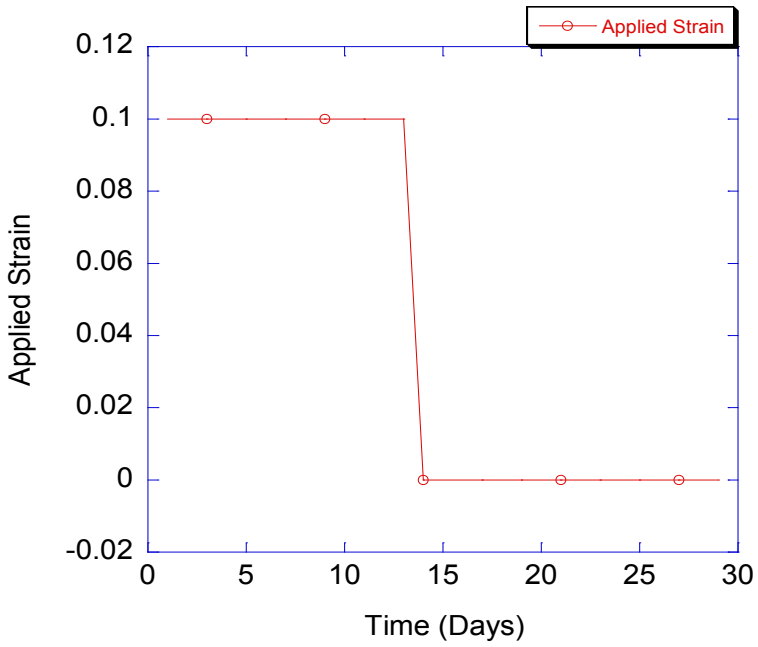

Figure 15 Virtual applied strain history.

According to the virtual applied strain history, the resultant apparent viscoelastic bulk stress predicted by the dissolution-formation viscoelastic model is shown in Figure 16, where bulk stress is defined as

$$Apparent\ Viscoelastic\ Bulk\ Stress = \frac{\sigma_{11} + \sigma_{22} + \sigma_{33}}{3}. \qquad (4.1)$$

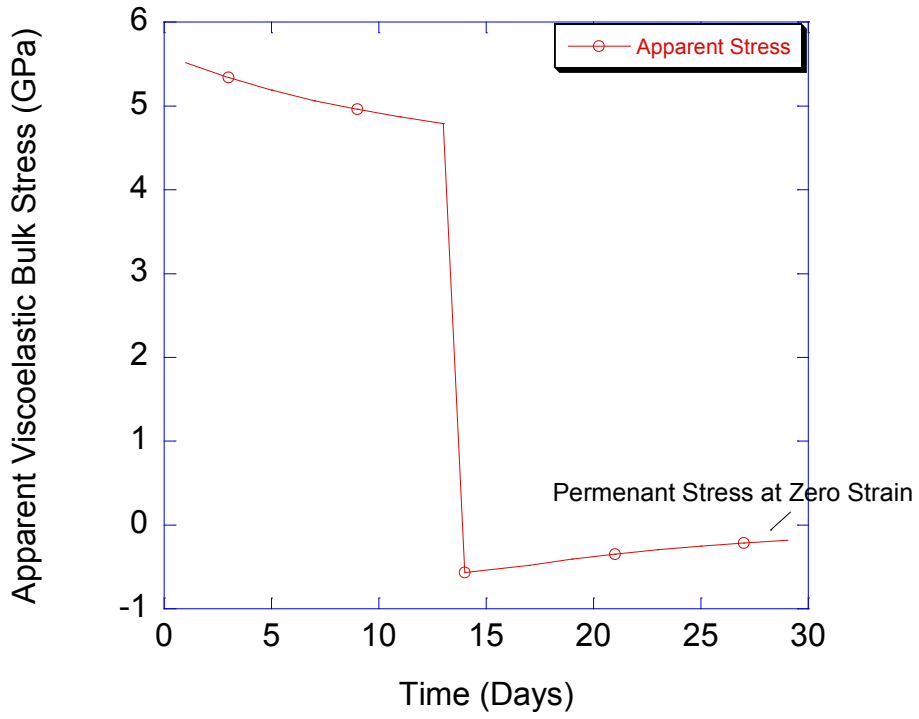

Figure 16 Response of aging viscoelastic composite material when a strain history shown in Figure 15 is applied.

It can be seen from Figure 16 that during the first 13 days, the composite showed relaxation under the constant controlled strain. At the 13th day, the strain was suddenly forced back to zero and this induced a sudden change in the response of the material. The apparent bulk stress inside the composite dropped down significantly.

One interesting thing is that, after the strain went back to zero, the stress in the material did not go back to zero but turned into an opposite value. That is if the composite was under compression during the first 13 days, it would suffer tension after the total stain was forced back to zero. Similarly, if the material suffered tension during the first 13 days, it would experience compression after the 13^{th} day. For materials that have new solids forming as time evolves, this makes sense because of the existence of free strain that arises out of the need for new phases to form in a stress-free state.

When a controlled strain was applied, the inherent components carried stresses and deformed. During the hydration process, some components inside the composite material dissolved and formed new components. These new components formed in the deformed configuration without carrying any stresses and this unstressed deformation was strain free. Thus, when the bulk strain of the microstructure was forced back to zero, there were opposite stresses from these components. After the sudden change of strain, the responses of the material went back to relaxation again and resulted in a permanent stress, as shown in Figure 16.

4.2 Viscoelasticity of C-S-H and Hydration

As discussed in Chapter 2, both viscoelasticity of C-S-H and the hydration process may play an important role in determining cement paste's viscoelastic properties. The dissolution-formation viscoelastic program makes the comparison of these two aspects possible and the comparison results are shown in detail in this section. Through

simulations of these two aspects, insight can be gained towards the mechanisms of viscoelastic properties of cementitious materials.

All the following results in this chapter are obtained through running the dissolution-formation viscoelastic program on the microstructures generated by THAMES on cement pastes of CCRL Proficiency Sample 168. The pastes are modeled as hydrated at 25℃ (298K) under sealed moisture conditions with different w/c ratios [27]. The sizes of these microstructures are 100 voxels by 100 voxels by 100 voxels, with each voxel 1 μm^3.

By applying controlled pure volumetric stress and pure shear stress on theses 100 voxels by 100 voxels by 100 voxels microstructures, apparent viscoelastic Bulk modulus and apparent vicoelastic Shear modulus can be obtained. With known apparent viscoelastic Bulk modulus and apparent vicoelastic Shear modulus, using Laplace Transform, apparent viscoelastic Young's modulus and apparent Poisson's ratio can be calculated.

Laplace Transform transforms expressions of apparent viscoelastic moduli from time-domain to frequency-domain. Meanwhile, Laplace transformation of the constitutive equation for linear isotropic viscoelasticity has the same form as that for linear elasticity when multiplying each parameter by *s*, where *s* is the changing variable in frequency-domain. Thus the relation between the four viscoelastic moduli in frequency-domain becomes:

$$s\bar{E} = \frac{9s\bar{K}s\bar{G}}{3s\bar{K} + s\bar{G}} \qquad s\bar{\nu} = \frac{3s\bar{K} - 2s\bar{G}}{2(3s\bar{K} + s\bar{G})}, \tag{4.2}$$

where $\bar{E}, \bar{K}, \bar{G}, \bar{\nu}$ are the Laplace transformation of apparent viscoelastic Young's modulus, apparent viscoelastic Bulk modulus, apparent viscoelastic Shear modulus and apparent viscoelastic Poisson's ratio. After obtaining the expressions for apparent viscoelastic Young's modulus and apparent viscoelastic Poisson's ratio in frequency-domain, they can be transferred back to time-domain for plotting.

Another way of calculating apparent viscoelastic Young's modulus and apparent viscoelastic Poisson's ratio with known apparent viscoelastic Bulk modulus and apparent viscoelastic Shear modulus is by using time steps. The continuous time in constitutive equation for linear viscoelasticity can be separated into discrete time steps and apparent viscoelastic Young's modulus and apparent viscoelastic Poisson's ratio at each time step can be calculated with known apparent viscoelastic Bulk modulus and apparent viscoelastic Shear modulus from all previous time steps.

These two different calculation methods give the same results.

4.2.1 Viscoelasticity of C-S-H

In this section, the viscoelastic Young's modulus for 1 day old cement paste and 28 day old cement paste were compared. The 1 day old viscoelastic Young's modulus was generated by utilizing the 1 day old microstructure and the 28 day old viscoelastic Young's modulus was generated by utilizing the 28 day old microstructure. In this way, only the viscoelastic properties of C-S-H dominated the viscoelastic Young's modulus results, and there was no microstructure change of the composite inside the microstructure for each set of the results.

Currently, due to experimental challenges, there are no sufficient data provided for modeling creep or relaxation of the C-S-H phase over several days, for example as the experiments shown in Figure 8, which last only 30s. Therefore, for simulation purposes, the viscoelastic Young's modulus of C-S-H was assumed to be $11.2 + 11.2e^{-0.2t}$, where t is in days and the Poisson's ratio of C-S-H was set to be a constant value of 0.2. The elastic properties of all other phases were taken from Table 1.

The responses of the cement paste are shown in Figure 17.

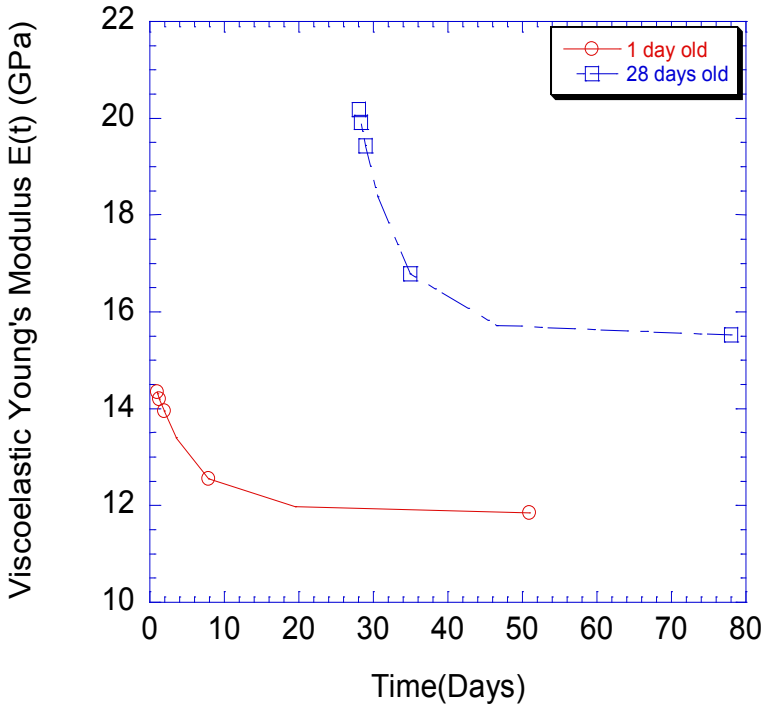

Figure 17 Viscoelastic Young's modulus for cement paste at different ages (1 day old and 28 day old) when dissolution and formation effect is not considered for the total viscoelastic behavior of cement paste. The viscoelastic Young's modulus of C-S-H is assumed to be $11.2 + 11.2e^{-0.2t}$*, where t is in days and the Poisson's ratio of C-S-H is assumed to be a constant value of 0.2. The elastic properties of all other phases were taken from Table 1.*

Figure 17 shows the viscoelastic response of cement paste due to the viscoelastic properties of C-S-H. According to the simulation results, a 28 day old cement paste would exhibit more relaxation than l day old cement paste. The reason behind the predicted higher relaxation for older materials is that as volume fraction of C-S-H increases with age, so do the magnitude of stresses transmitted by C-S-H. Therefore, if C-S-H is the only viscoelastic phase in cement paste, more C-S-H results in more relaxation or creep. However, these predictions are in contrast to experimental results, which shows that there is more relaxation in younger cement paste than in older materials[37, 38]. Thus, it appears that the inherent viscoelasticity of C-S-H cannot be the primary mechanism of cement paste viscoelasticity because its confliction with experimental evidence of reduction in viscoelastic relaxation with age. A further discussion of this issue will be shown in section 4.2.3.

The solidification theory considers the forming of new phases but it does not include the dissolution of phases. The solidification theory implies that hydration products are non-aging viscoelastic materials and the amount of solidified products increases with time, as shown in Figure 4. The load bearing materials increase with time as a result of solidification and deposition of hydration products. However, in reality, formation of new load bearing phases requires dissolution of existent loading bearing phases, which is not included in the solidification theory. As a result of increasing amount of load bearing phases in the solidification theory, the stress carried by new loading bearing phases is gradually decreasing, and cement paste at later age will show less relaxation because of the decreasing inherent averaged stress, which is different

from the results shown in Figure 17. Detailed comparison between the solidification theory and the dissolution-formation viscoelastic program will be shown in later sections.

4.2.2 Viscoelasticity of C-S-H and Dissolution

When examining both effect of dissolution of load bearing phases and C-S-H viscoelasticity, different results were shown from Figure 17.

As mentioned in chapter 3, discrete time steps were used in the dissolution-formation viscoelastic program, time steps representing different ages of the whole composite material. When time equals 1 day in the program, microstructure at 1 day old was input into the program. As program runs, time steps increases. When time increases to 1.5 day, the 1.5 day old microstructure was input into the program for analysis instead of the 1 day old microstructure. In the end, all microstructures would be analyzed and the progress of dissolution of unhydrated phases can be accounted for by applying these different microstructures. 0.4 w/c microstructures from 1 day to 4.5 days were used to generate Figure 18, which shows the predicted results when examining both effect of dissolution of load bearing phases and inherent C-S-H viscoelasticity.

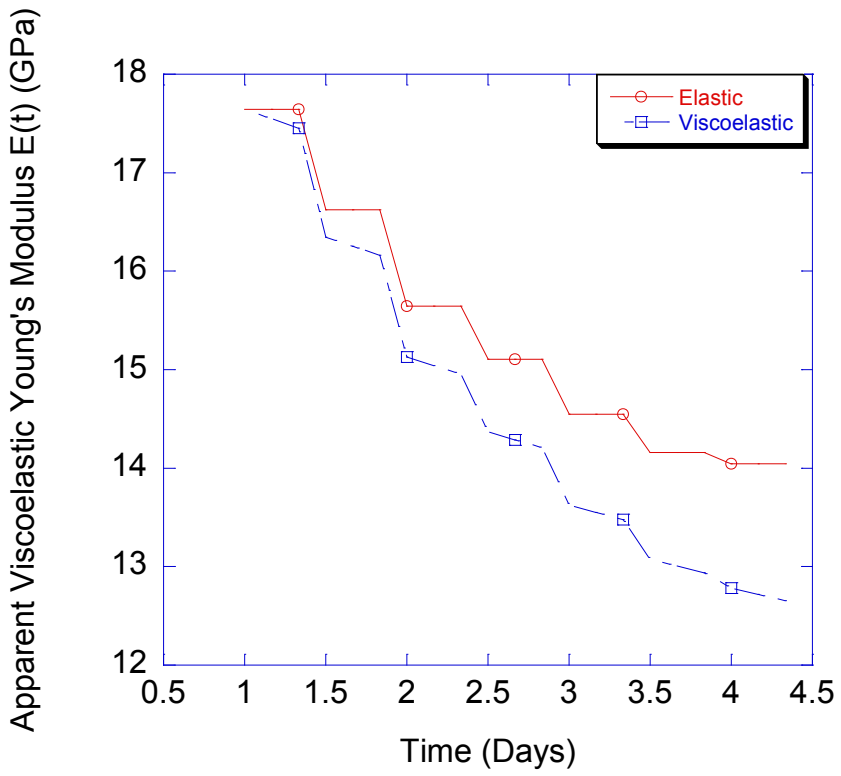

Figure 18 Response of cement paste with an assumed viscoelastic Young's modulus of C-S-H comparing to the response of cement paste when C-S-H is considered elastic. The viscoelastic Young's modulus of C-S-H is assumed to be $11.2 + 11.2e^{-0.2t}$*, where t is in days and the Poisson's ratio of C-S-H is assumed to be a constant value 0.25. The elastic properties of all other phases were taken from Table 1.*

In Figure 18, the solid line shows the response when C-S-H is considered to be an elastic material, and the dashed line shows the response when C-S-H is treated as viscoelastic. The C-S-H phases were assumed to be the only viscoelastic phase in cement paste with the assumed viscoelastic Young's modulus $11.2 + 11.2e^{-0.2t}$, where t is in days, and the Poisson's ratio was set to be a constant 0.25.

From Figure 18, it can be seen that, the main trend of the viscoelasticity of cement paste is partially determined by the dissolution effect as well as the

viscoelasticity of C-S-H. The relaxation effect caused by C-S-H viscoelasticity increases with time.

4.2.3 Mechanisms of Viscoelasticity of Cement Paste

As mentioned, if the C-S-H phase shows a significant amount of relaxation, viscoelasticity of C-S-H contributes to the viscoelasticity of the whole microstructure. However, from the simulation results of the dissolution-formation viscoelastic program, another main reason leading to the viscoelastic property of cement paste is the dissolution of load bearing phases.

Another finding by the dissolution-formation viscoelastic program is that even if viscoelasticity of C-S-H can contribute to the viscoelasticity of the whole microstructure, it cannot be the main mechanism leading to the viscoelastic behavior of cement paste. The only way that C-S-H viscoelastic deformation could be the dominant mechanism would be that if the dominant aging mechanism were inherent aging of C-S-H rather than the hydration process. That is, the viscoelastic properties of C-S-H would need to be dramatically evolved with the age of C-S-H particles.

Further experiments are needed to clearly determine the properties of C-S-H over a broader time scale.

4.3 Predicted Effect of Different Loading Ages

4.3.1 Apparent Viscoelastic Young's Modulus

It is known that the responses of aging viscoelastic materials depend significantly on loading histories. If loads are applied at different ages, the responses would be different. This section shows the predicted responses of cement paste at different loading ages (1 day and 7 days) by the dissolution-formation viscoelastic program.

Figure 19 shows the predicted apparent viscoelastic Young's modulus of 0.40 w/c paste with different loading ages of 1 day and 7 days. All phases in the composite were assumed to be elastic and the results were generated using the microstructures from 1 day to 56 days. For loading age of 1 day, microstructures from 1 day to 56 days were used and boundary conditions were applied starting from the 1^{st} day microstructure. For loading age of 7 days, microstructures from 7 days to 56 days were used and boundary conditions were applied starting from the 7^{th} day microstructure. The elastic properties of the components inside the composite material were taken from Table 1, including C-S-H. In this way, only the effect of dissolution was considered, and the relaxation was due entirely to dissolution effects.

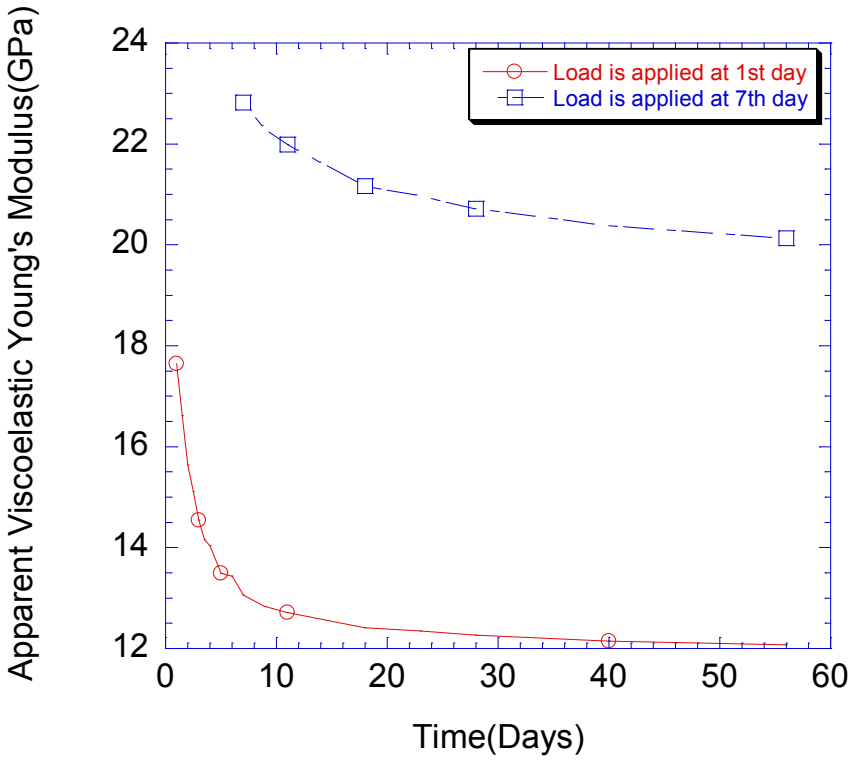

Figure 19 Apparent viscoelastic Young's modulus of 0.40 w/c paste when loaded at different ages. In this graph, apparent viscoelasticity is considered to occur strictly due to dissolution of load bearing phases

Cement paste will generate a higher elastic Young's modulus when loaded at a later age. This is because, when loaded at a later age, as hydration process goes on, more and more water and cement react into C-S-H and CH, which is also called the process of solidification. During this process, the cement paste becomes stiffer and stiffer because of increasing interparticle connections and reduced porosity.

To examine the effect of loading ages on rate of relaxation, data from Figure 19 were normalized by instantaneous apparent viscoelastic Young's modulus to get Figure 20.

4.3.2 Rate of Relaxation

Figure 20 shows the effect of different loading ages on normalized apparent viscoelastic relaxation.

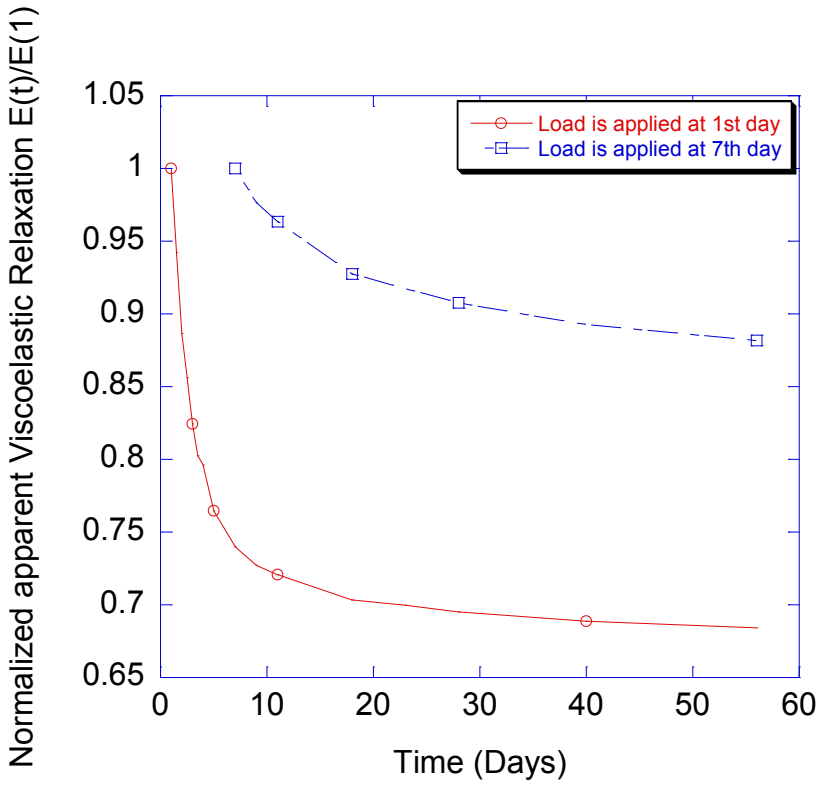

Figure 20 Effect of initial loading age (1st day and 7th day) on normalized apparent viscoelastic relaxation, where apparent viscoelastic Young's modulus E(t) was normalized by instantaneous apparent viscoelastic Young's modulus. In this graph, viscoelasticity is considered to occur strictly due to dissolution of load bearing phases. The dissolution effect is able to account for aging effect of viscoelastic behavior.

From Figure 20, it can be seen that cement paste loaded at earlier age shows larger relaxation, which is consistent with experimental results. As mentioned, dissolution of load bearing phases is substantial to viscoelasticity of cement paste, and

from Figure 20, the dissolution effects is able to account for the aging effect of viscoelastic behavior of cement paste. As the hydration rate slows as specimens age, the rate of dissolution also decreases, leading to the aging effect of viscoelasticity shown in Figure 20.

4.4 Predicted Effect of Different w/c

4.4.1 Apparent Viscoelastic Young's Modulus

Water/cement ratio also has big influences on viscoelastic behavior of cementitious materials. Microstructures for different w/c can be analyzed by the dissolution-formation viscoelastic program. In this section, all phases in the composite material were also assumed to be elastic and only the dissolution effect is examined. The results were generated using the microstructures from 1 day to 56 days with different w/c and the elastic properties of the components inside the composite material were taken from Table 1.

After running the dissolution-formation viscoelastic program on these microstructures, the results of apparent viscoelastic Young's modulus considering loading ages of 1 day and 7 days are shown in Figure 21 and Figure 22.

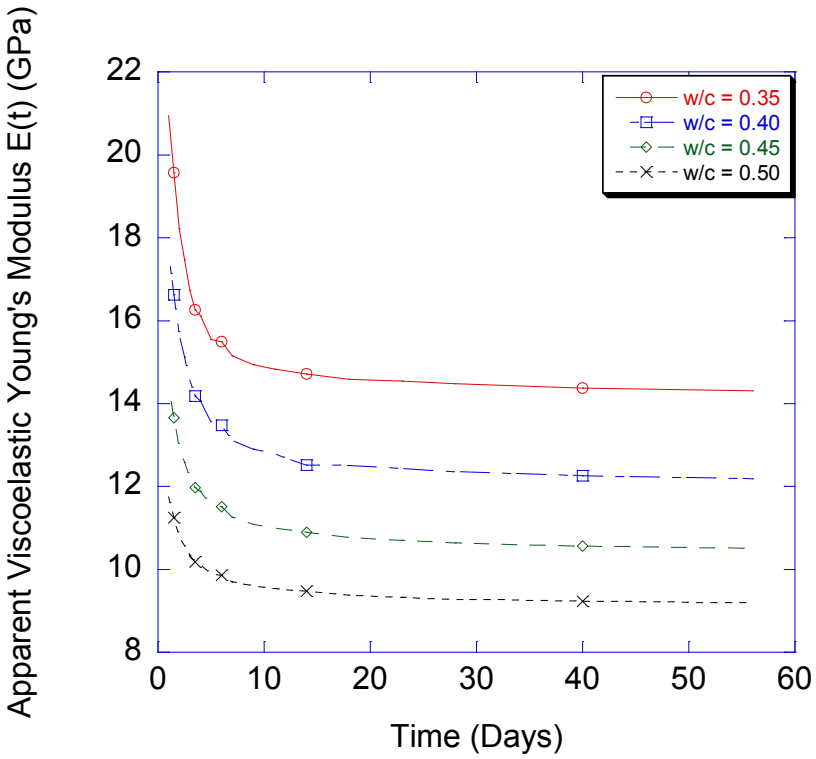

Figure 21 Apparent viscoelastic Young' modulus for different w/c at loading age of 1 day. Viscoelasticity is considered to occur strictly due to dissolution of load bearing phases.

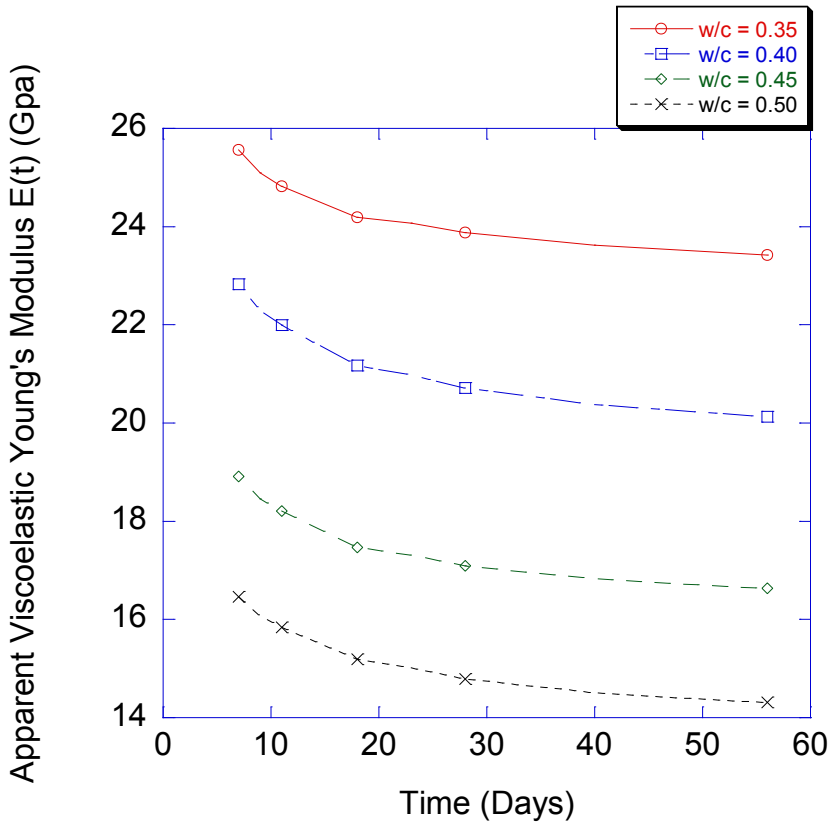

Figure 22 Apparent viscoelastic Young' modulus for different w/c at loading age of 7 days. Viscoelasticity is considered to occur strictly due to dissolution of load bearing phases.

As w/c goes up, the initial elastic Young's modulus goes down, which is reasonable and agrees with experiments. Value of apparent viscoelastic Young's modulus reduces at a decreasing rate with time.

To get a clear view of the effect of w/c on rate of relaxation, data from Figure 21 and Figure 22 were normalized by instantaneous elastic Young's modulus to get Figure 23 and Figure 24.

4.4.2 Rate of Relaxation

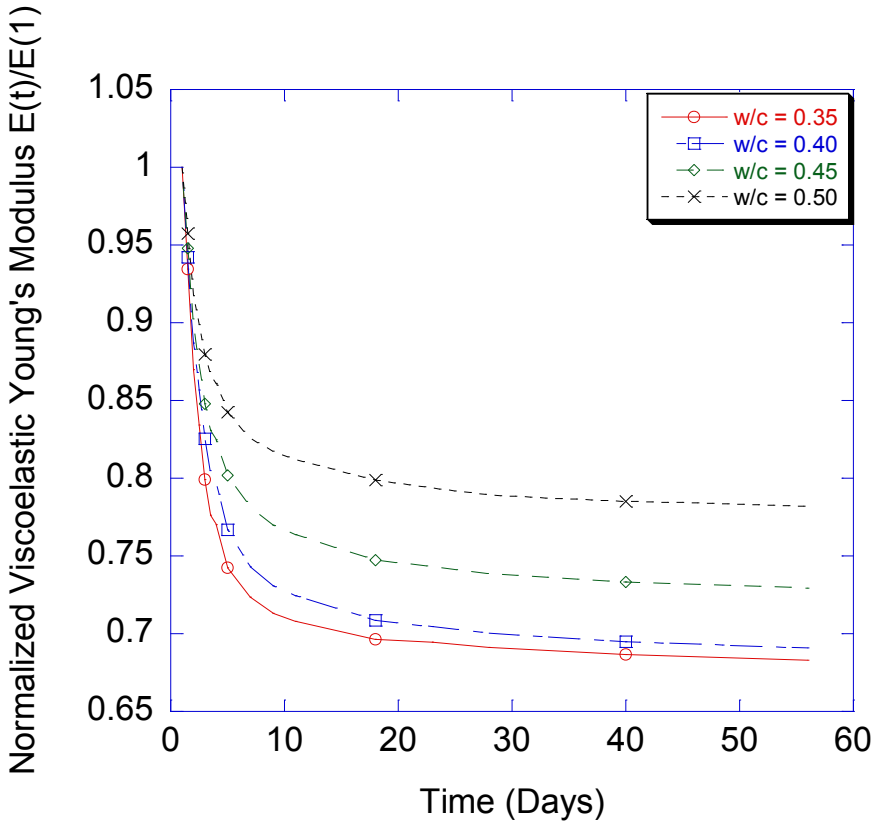

Figure 23 Effect of different w/c on normalized apparent viscoelastic relaxation at loading age of 1 day, where apparent viscoelastic Young's modulus E(t) was normalized by instantaneous elastic Young's modulus. Viscoelasticity is considered to occur strictly due to dissolution of load bearing phases.

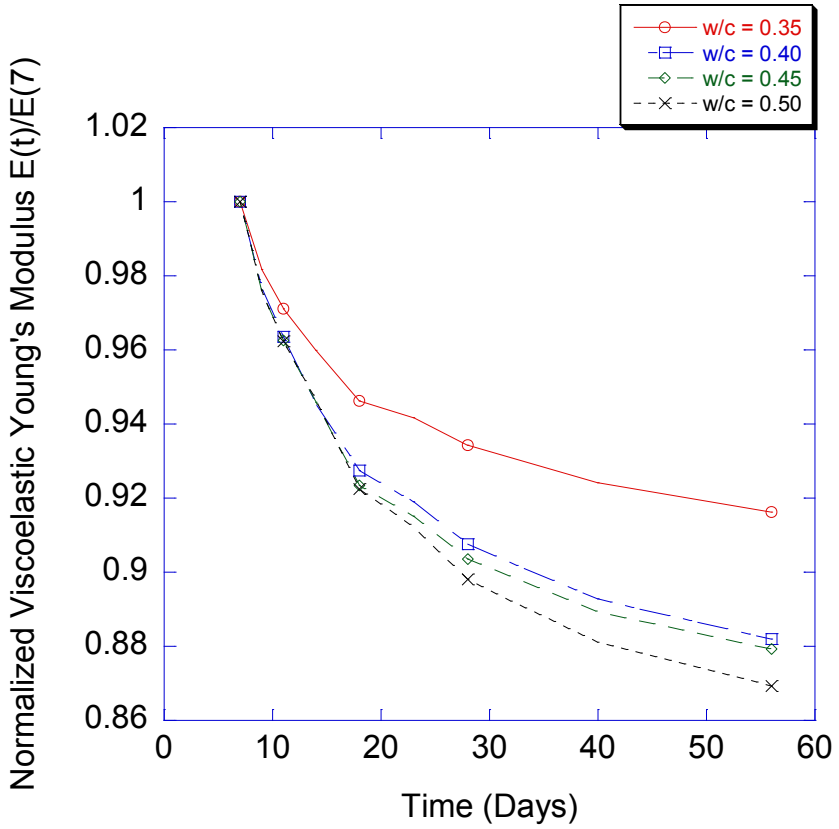

Figure 24 Effect of different w/c on normalized apparent viscoelastic relaxation at loading age of 7 days, where apparent viscoelastic Young's modulus E(t) was normalized by instantaneous elastic Young's modulus. Viscoelasticity is considered to occur strictly due to dissolution of load bearing phases.

Figure 23 and Figure 24 show the results of normalized viscoelastic relaxation with different w/c.

First, from these two figures, for all w/c, cement at younger ages show higher rate of relaxation because the rate of hydration (and thus rate of dissolution) is higher at 1 day than at 7 days. As discussed in section 4.3, dissolution of load bearing phases has a great impact on viscoelasticity of cement paste; viscoelastic properties of cement paste are closely related to rate of hydration reaction.

Second, w/c has different influences on rate of relaxation at different loading ages.

For younger specimens with a loading age of 1 day, according to W. Vichit-Vadakan and George W. Scherer's experiments [37], lower w/c generates a higher relaxation rate, as shown in Figure 25：

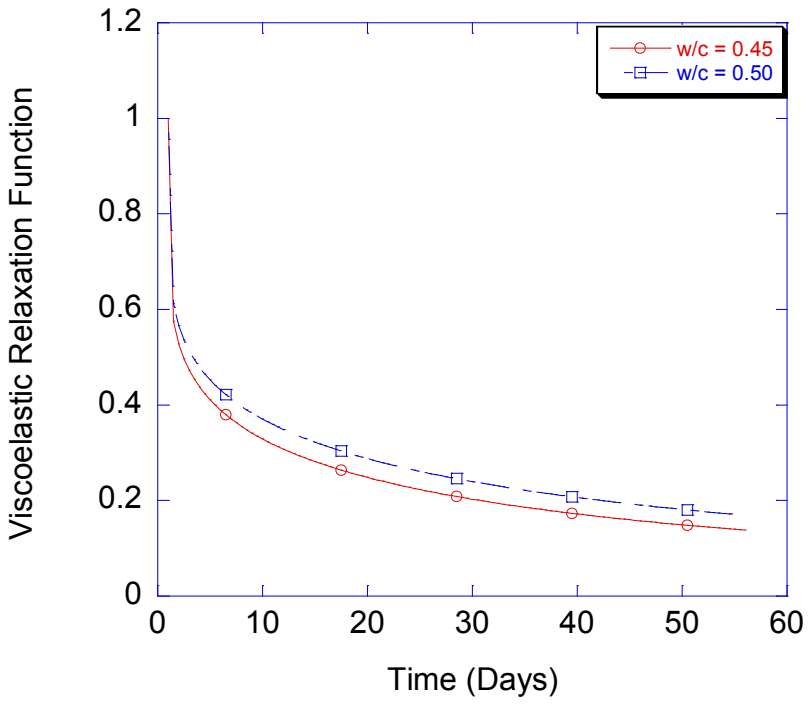

Figure 25 Effect of different w/c on viscoelastic relaxation at loading age of 1 day. The curve is gained through plotting the viscoelastic relaxation function in [37] and it is found that the relaxation is faster for cement pastes with lower w/c.

Figure 23 gives consistent results. For 1 day old cement paste, hydration reaction is very active and thus it is the dominate factor affecting the relaxation rate. Because at

early age, lower w/c gives higher hydration reaction rate [12], faster relaxation is generated.

For older specimens, however, the rate of relaxation is greater for cement pastes with higher w/c. There are two reasons leading to this:

First, under the age of 7 days, cement pastes with higher w/c can have a higher hydration rate. At this point, amount of water becomes the factor limiting the rate of hydration, and higher w/c can provide more water, resulting in higher hydration rate.

Second, for cement pastes with lower w/c, when load bearing phases dissolve, inherent stress and strain are redistributed to larger volume of solidified phases. This lead to lower relaxation rate.

In conclusion, lower w/c generates higher relaxation rate for younger specimens, but lower relaxation rate for older specimens.

4.4.3 Other Important Parameters

Figure 26 and Figure 27 show the predicted results of apparent viscoelastic Bulk modulus, apparent viscoelastic Shear modulus and apparent viscoelastic Poisson's ratio at the two loading ages. Also, when generating the data for Figure 26 and Figure 27, all phases in the composite material were assumed to be elastic and only the dissolution effect was examined. The results were generated using the microstructures from 1 day to 56 days with different w/c and the elastic properties of the components inside the composite material were taken from Table 1.

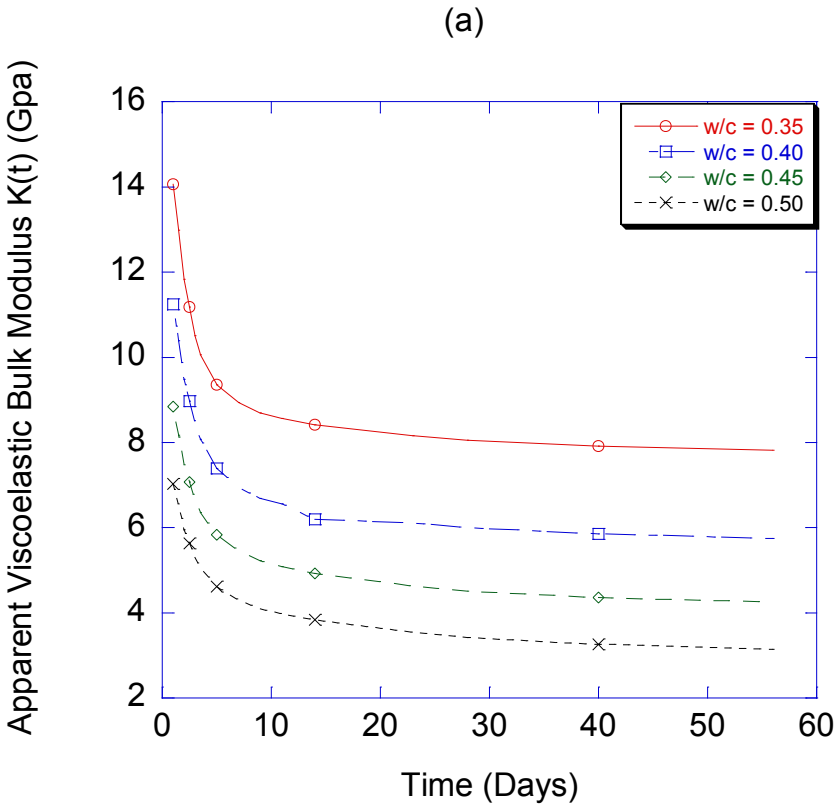

Figure 26 Apparent viscoelastic (a) Bulk modulus, (b) Shear modulus and (c) Poisson's ratio results for different w/c at loading age of 1 day. Viscoelasticity is considered to occur strictly due to dissolution of load bearing phases.

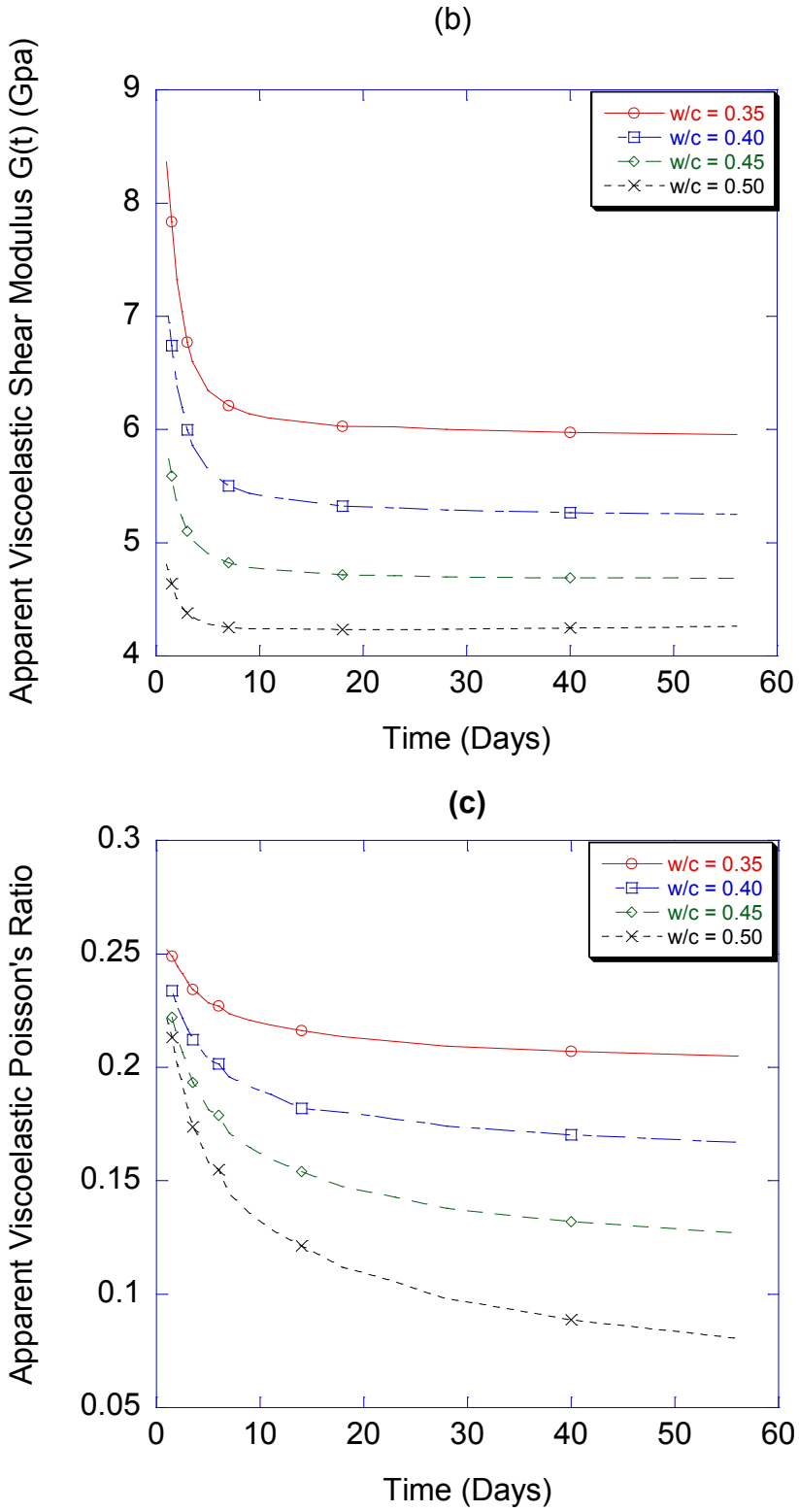

Figure 26 Continued.

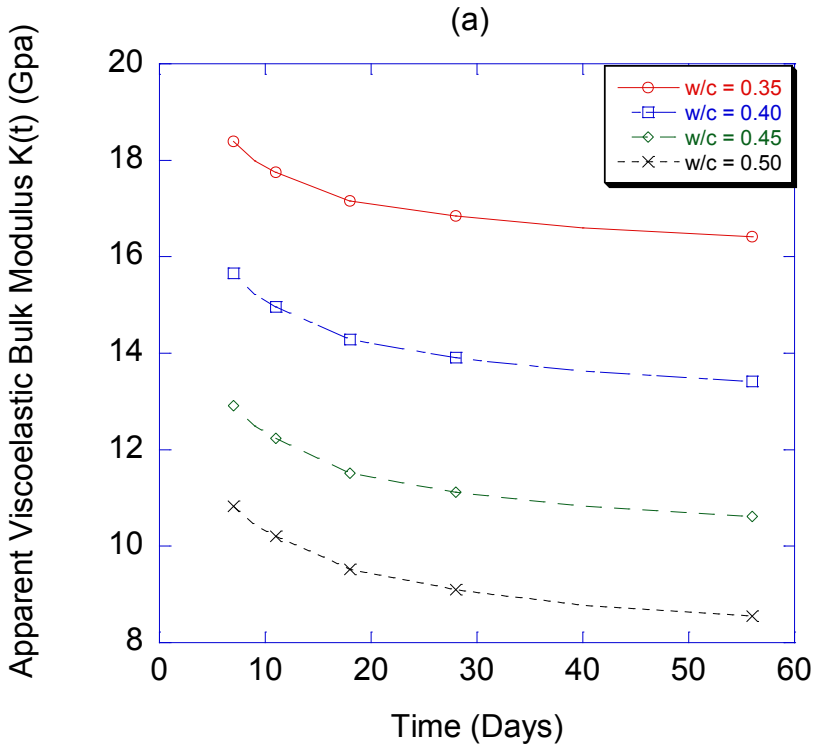

Figure 27 Apparent viscoelastic (a) Bulk modulus, (b) Shear modulus and (c) Poisson's ratio results for different w/c at loading age of 7 days. Viscoelasticity is considered to occur strictly due to dissolution of load bearing phases.

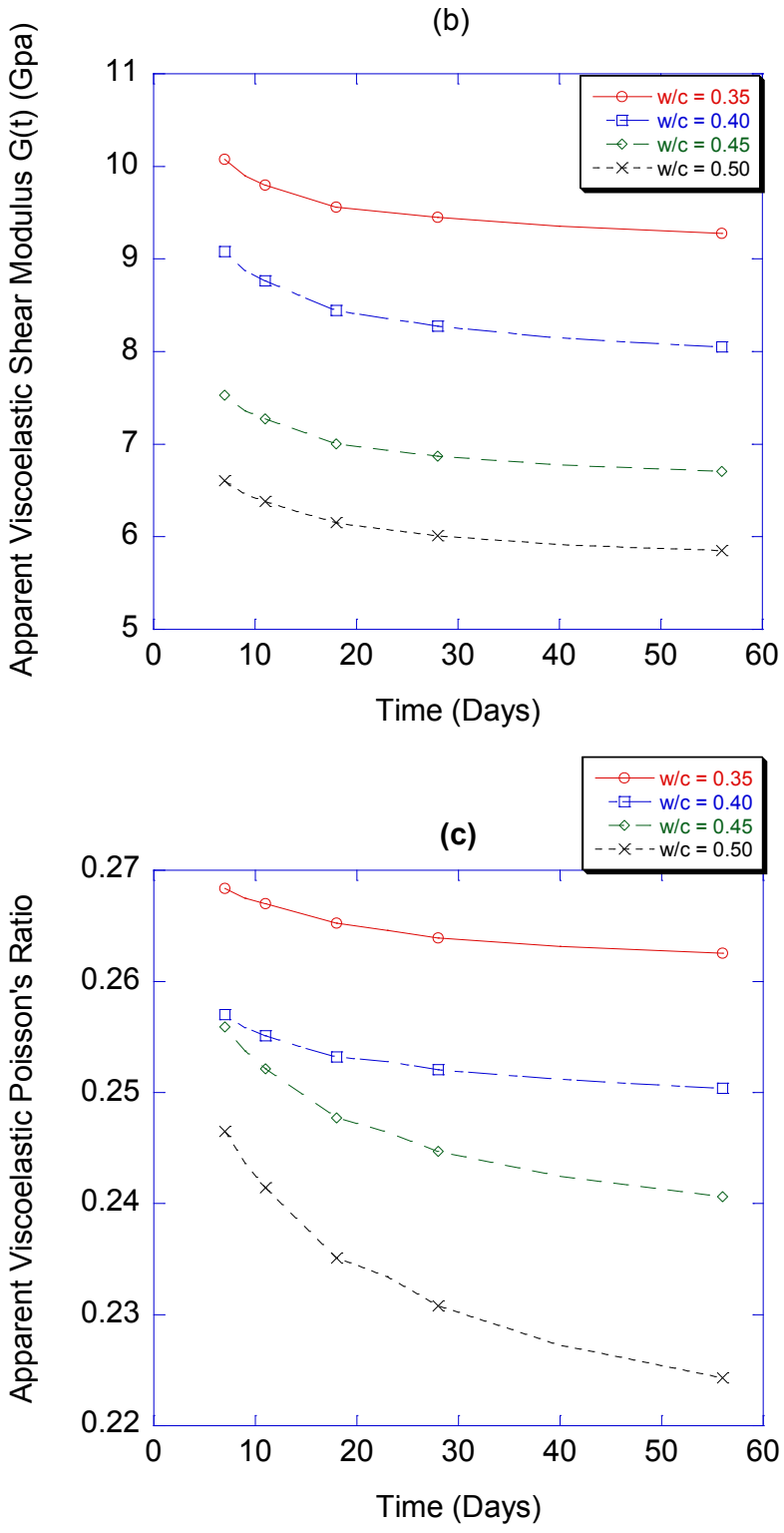

Figure 27 Continued.

From Figure 26 and Figure 27, apparent viscoelastic Bulk modulus and apparent viscoelastic Shear modulus show the same relaxation trend as apparent viscoelastic Young's modulus. For apparent viscoelastic Bulk modulus and apparent viscoelastic Shear modulus, as w/c goes down, because the composite becomes stiffer, the initial elastic values for these parameters go up. As time passes by, all these parameters become smaller due to the relaxation of cement paste. Because of the decreasing hydration reaction rate (rate of dissolution), the rate of decrease in these parameters reduces with time. Later loading age shows higher initial elastic values of these parameters. All these results agree with experiments.

However, for apparent viscoelastic Poisson's ratio, it can be seen that their initial values for different w/c and for different loading ages do not differ as much as for apparent viscoelastic Bulk modulus and apparent viscoelastic Shear modulus.

Figure 26 and Figure 27 also show that values of apparent viscoelastic Poisson's ratio decrease with time. One reason for this is that, as specimens age, stresses are gradually transferring from unhydrated phases (such as C_3S) to hydrated phases (such as C-S-H). Hydrated phases have smaller values of Poisson's ratio than unhydrated phases, resulting in the decrease in Poisson's ratio. Another main reason leading to the decrease in Poisson's ratio is the dissolution-formation process of cement paste. Unhydrated phases dissolve and new phases form in stress-free state in the deformed configuration. These stress-free solidified phases have free strain, and they constraint surrounding phases from deforming, causing a decrease in Poisson's ratio. Based on this, more intense dissolution-formation process would induce larger change in Poisson's ratio.

This is the reason for larger decrease in Poisson's ratio caused by higher w/c. And this is also the reason why an earlier loading age can give a much larger change in Poisson's ratio than an older loading age, because an earlier loading age will make the specimen sustain much more intense dissolution-formation process after load is applied.

Through running the dissolution-formation viscoelastic program, prediction of some parameters which are difficult to measure through actual experiments, such as apparent viscoelastic Poisson's ratio, becomes achievable. Prediction of stresses (volumetric stress and deviatoric stress) carried by different phases (solidified phases and unhydrated phases) also becomes achievable.

4.5 Predicted Stress Distribution

4.5.1 Volumetric Stress and Deviatoric Stress

Volumetric stress is the stress tending to change the volume of the stressed body, and its magnitude is given by

$$p = \frac{\sigma_{11} + \sigma_{22} + \sigma_{33}}{3}, \tag{4.3}$$

where p is the volumetric stress, and σ_{11}, σ_{22}, σ_{33} are normal stresses in three directions.

Deviatoric stress is the stress which tends to distort the stressed body. The stress deviator tensor is given by

$$[s_{ij}] = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix} = \begin{bmatrix} \sigma_{11} - p & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} - p & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} - p \end{bmatrix}, \tag{4.4}$$

where s_{ij} is deviatoric stress tensor, and σ_{ij} is the stress tensor. A scalar measure of deviatoric stress magnitude is

$$s = \sqrt{\frac{1}{2} s_{ij} s_{ji}}, \tag{4.5}$$

where *s* is the deviatoric stress and *i, j* = 1, 2, or 3.

The three invariants of stress tensor σ_{ij} are

$$I_1 = \sigma_{11} + \sigma_{22} + \sigma_{33}, \tag{4.6}$$

$$I_2 = \frac{1}{2}\left(\sigma_{ii}\sigma_{jj} - \sigma_{ij}\sigma_{ji}\right), \tag{4.7}$$

$$I_3 = \sigma_{11}\sigma_{22}\sigma_{33} + 2\sigma_{12}\sigma_{23}\sigma_{31} - {\sigma_{12}}^2\sigma_{33} - {\sigma_{23}}^2\sigma_{11} - {\sigma_{13}}^2\sigma_{22}, \tag{4.8}$$

where I_n are the nth invariant, and *i, j* = 1, 2, or 3. From eq. (4.3) and eq. (4.6), it can be seen that volumetric stress and the first invariant are related to each other. Meanwhile, from eq. (4.4), eq. (4.5) and eq. (4.7), one finds:

$$s^2 = \frac{1}{3}{I_1}^2 - I_2. \tag{4.9}$$

Thus, both volumetric stress and deviatoric stress are closely related to the three invariants of the stress tensor, and their changes with time can represent the corresponding changes in the stress.

Figure 28 shows the predicted results of normalized volumetric stresses and deviatoric stresses carried by hydrated phases and unhydrated phases at different ages.

Volumetric stresses are normalized by total volumetric stress carried by the whole microstructure and deviatoric stresses are normalized by total deviatoric stress carried by the whole microstructure. By summing up all the normalized stresses (volumetric stress and deviatoric stress) carried by voxels of hydrated phases or by voxels of unhydrated phases, Figure 28 can be obtained. 0.4 w/c microstructures at different ages were used in the program, while assuming all phases in the composite microstructure were elastic with elastic properties shown in Table 1. The data were gathered considering a loading age of 1 day.

For both volumetric and deviatoric stress,

$$Normalized\ \sigma(t) = \frac{\sum \sigma_{hydrated\ /unhydrated}\ (t)}{<\sigma(t)>}, \tag{4.10}$$

where $Normalized\ \sigma(t)$ is normalized volumetric or deviatoric stress at time *t*, $\sum \sigma_{hydrated\ /unhydrated}\ (t)$ is the summed volumetric or deviatoric stress over hydrated or unhydrated voxels at time *t*, and $<\sigma(t)>$ is the integrated volumetric or deviatoric stress over the whole volume of the microstructure at time *t*.

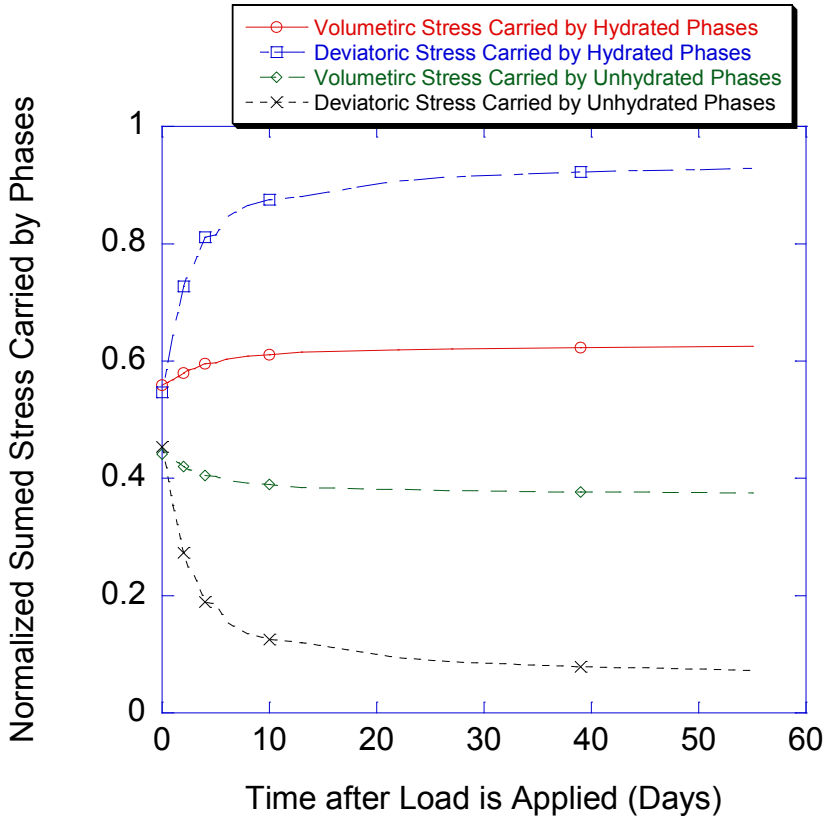

Figure 28 Normalized volumetric stress and deviatoric stress carried by hydrated phases and unhydrated phases inside cement paste microstructure under loading age of 1 day. For the summed volumetric and deviatoric stresses carried by hydrated phases and unhydrated phases at time t, they are normalized by total volumetric stress or total deviatoric stress carried by the whole microstructure at time t respectively.

From Figure 28, it can be seen that, during the dissolution and formation process, stresses are gradually transferred from unhydrated phases to solidified phases. Both normalized volumetric stress and normalized deviatoric stress carried by unhydrated phases are decreasing with age, while for hydrated phases, these two stresses are increasing. Meanwhile, more deviatoric stress is transferred than volumetric stress. With the existence of free strain, under a strictly strain controlled condition, volume changes

of hydrated voxels are constrained, as well as the volume changes of surrounding voxels. On the other hand, hydrated phases form in a deformed irregular non-cubic configuration, more shape changes in hydrated phases is reasonable.

4.5.2 Comparison between Dissolution-Formation Viscoelastic Model and Solidification Theory

As mentioned in previous sections, the solidification theory only considers the forming of new phases without considering the dissolution of load bearing phases. As a result of the increasing amount of load bearing phases, the average stress carried by solidified components is gradually getting smaller. However, from Figure 28, the results from the dissolution-formation viscoelastic model give an increase in the stress carried by solidified phases with age. Comparison results between the new model and the solidification theory is necessary.

When Bazant proposed the solidification theory, the specimen was analyzed under a uniaxial load, as shown in Figure 4. However, as discussed in the previous section, volumetric stress change and deviatoric stress change can represent the stress change of the microstructure, and comparing to the case of uniaxial stress, volumetric stress and deviatoric stress should give the same trends. Thus, volumetric stress and deviatoric stress are analyzed in this section. Detailed results are shown in Figure 29 and Figure 30. One thing to note is that, to show clear comparison, only the phases that solidify after load is applied are considered as solidified phases in Figure 29 and Figure 30. The stresses in Figure 29 and Figure 30 are normalized by $< \sigma(t_0) >$, the integrated

volumetric or deviatoric stress over the whole volume of the microstructure at time *t=0,* when load is applied.

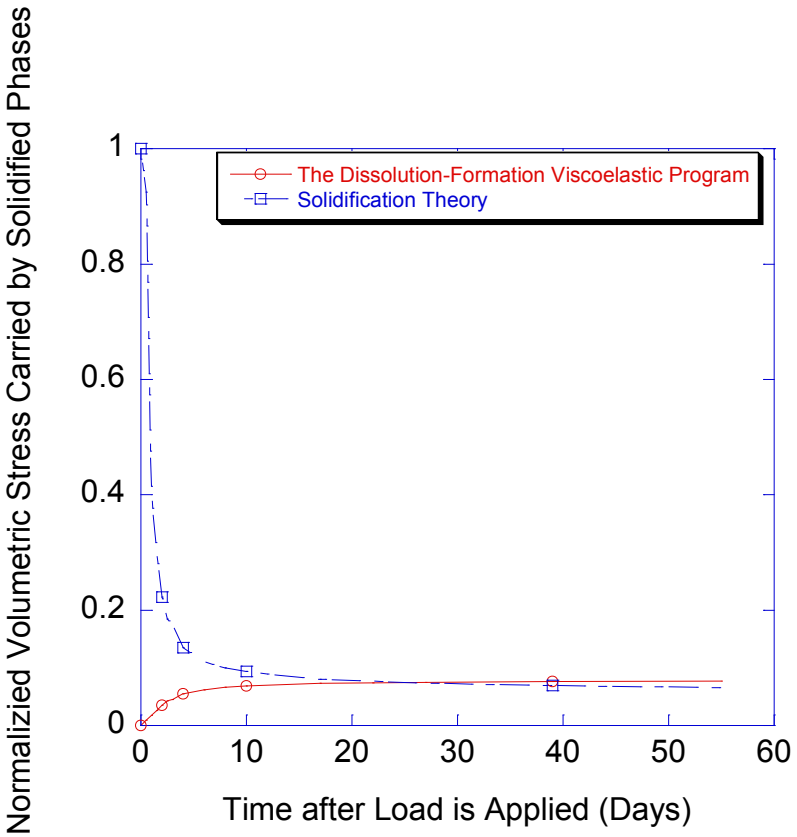

Figure 29 Comparison between the dissolution-formation viscoelastic model and the solidification theory on normalized volumetric stress carried by solidified phases. The stresses are normalized by instantaneous total volumetric stress carried by the microstructure. Only the phases that solidify after load is applied are considered as solidified phases in this figure.

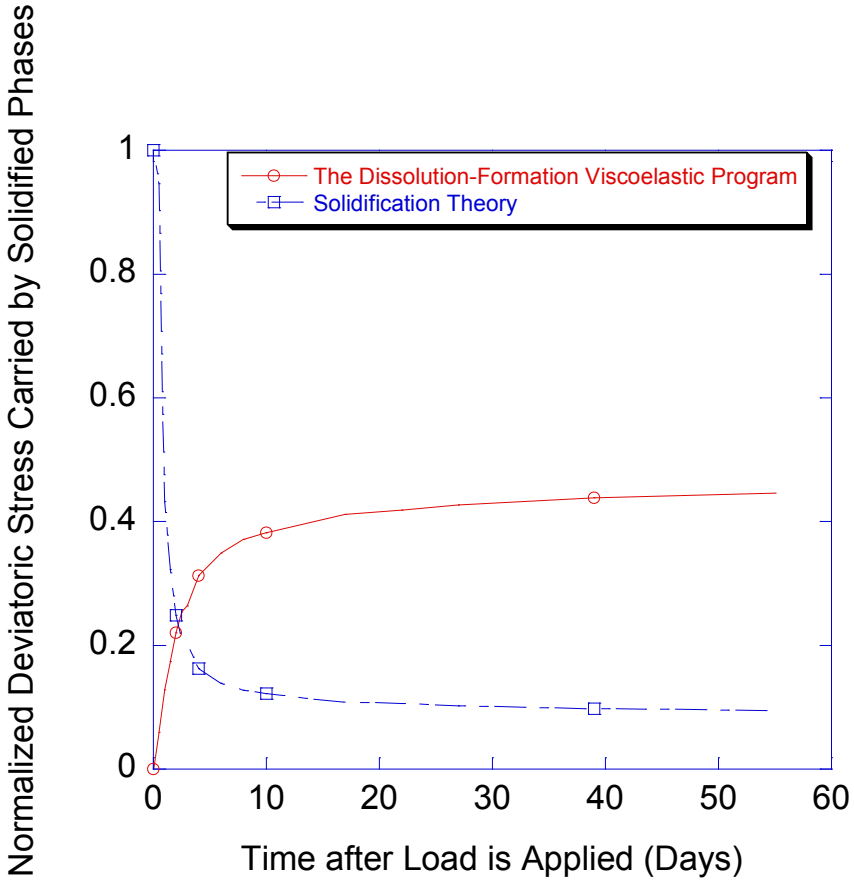

Figure 30 Comparison between the dissolution-formation viscoelastic model and the solidification theory on normalized deviatoric stress carried by solidified phases. The stresses are normalized by instantaneous total deviatoric stress carried by the microstructure. Only the phases that solidify after load is applied are considered as solidified phase in this figure.

In Figure 29 and Figure 30, the normalized stress carried by the solidified products for the solidification theory is calculated through

$$\sigma(t) = \frac{\sigma_{total}\,(t)}{v(t)}, \tag{4.11}$$

where $v(t)$ is the aging function depending on volume fraction of solidified phases, $\sigma(t)$ is the normalized averaged total stress carried by solidified phases at time *t*, and $\sigma_{total}\,(t)$

is the normalized total external stress, either volumetric stress or deviatoric stress, predicted by the dissolution-formation viscoelastic program at time *t*.

The curves of stresses carried by solidified phases for the dissolution-formation viscoelastic program are obtained through summing up all the normalized stresses carried by the phases that solidify after load is applied. The same microstructures are used as in Figure 28.

From Figure 29 and Figure 30, it can be seen that, in the solidification theory, averaged stress carried by solidified phases is decreasing with time; in the dissolution-formation viscoelastic program, the total stress carried by solidified phases is increasing with time. The reason that the solidification theory can predict the aging effect of cement paste is due to the decreasing average stress carried by solidified phases, while in reality, the stresses carried by solidified phases, such as C-S-H, are increasing, as predicted by the dissolution-formation viscoelastic program.

CHAPTER V

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The aging viscoelastic properties of cement-based materials are important for accurately predicting stress and strain fields in cement-based infrastructure. However, there are no fundamental models describing the viscoelastic behavior adequately. This is because cement-based materials are composite materials with random matrix arrangement and the internal chemical components of cement-based materials are changing during the hydration process. To effectively solve this problem, an analytical constitutive modeling of cement paste using the finite element method is discussed in this thesis.

The object of the model is to predict viscoelastic constitutive properties based on cement characteristics and link observed viscoelastic behavior with evolving microstructure linked to the hydration process. After developing the model, through running virtual experiments, several results can be found.

First, this dissolution-formation viscoelastic model gives reasonable results for various cases, while the solidification theory predicts the aging process based on two simplifications. The solidification theory assumes that solidified phases are the only phases that carry stress and the stresses in unhydrated phases are zero. Meanwhile, without considering the dissolution of load bearing phases during the hydration process,

as hydration reaction goes on, the inherent average stress carried by solidified phases decreases. This is the reason the solidification theory predicts the aging effect of cement paste, while in reality, the stress carried by unhydrated phases are not zero, and as hydration goes on, the stresses in solidified components gradually increase. The dissolution-formation viscoelastic program makes no such simplifications and accounts for the aging effect by the dissolution of load bearing phases.

Second, and also the most important finding is that the apparent viscoelastic behavior due to dissolution of load bearing phases is substantial. The dissolution process occurring during the hydration reaction can change the stress distribution inside cementitious materials significantly, resulting in apparent viscoelastic behavior of the whole cementitious material.

From the results obtained from the dissolution-formation viscoelastic program, there is a possibility that the main contribution to the viscoelastic properties of cement paste is the hydration process instead of the viscoelastic properties of the C-S-H phase. One reason is that experimental results indicate that creep of C-S-H on the nanometer length scale is less than what is observed on the micrometer length scale, indicating that there are additional creep mechanisms present beyond intrinsic C-S-H creep. C-S-H may not exhibit as strong of viscoelastic behavior as previously thought. Another reason is that, if viscoelasticity of C-S-H is the main mechanism leading to viscoelastic behavior of cement paste, simulations indicate that older materials would exhibit more creep than younger materials, which conflicts with myriad experimental data. This finding requires new thoughts towards the mechanisms of viscoelasticity of cementitious materials.

5.2 Future Work

For future work, first, the accuracy of the dissolution-formation viscoelastic model still needs to be checked through experiments. Relaxation tests can be carried out on cement pastes and comparison between the results from the program and the results from experiments is required. After comparing the experimental results for different w/c at different loading ages with the results from the dissolution-formation viscoelastic program, the dissolution-formation viscoelastic model should be further modified if large errors show up.

Second, more virtual experiments can be carried out to deeply examine different parameters that are difficult to be measured through realistic experiments, such as the apparent viscoelastic Poisson's ratio. The change of these parameters to different characteristics of cement paste, or to different loading histories is of great interest and can help better our understanding of the mechanisms of deformation in cementitious materials.

Third, although the computational program indicates that the dissolution effect is able to account for the aging effect of viscoelasticity, hydration slows dramatically after an age of about 28 days (under typical curing conditions) while creep can continue long after 28 days. Thus, there must be other mechanisms of viscoelasticity other than C-S-H viscoelasticity and dissolution of unhydrated cement grains. Meanwhile, based on experimental data, this program highly underpredicts the relaxation inside cement paste. Thus, deeper insight and new ideas towards mechanisms of cementitious materials

viscoelasticity are required. One possible mechanism is the dissolution caused by thermodynamic equilibrium. Thermodynamic defines which constituents are stable under a particular set of conditions — including the state of stress and rate of deformation. Therefore, for multi-phase materials exposed to mechanical loading, to accurately predict its microstructure evolution (or dissolution of unhydrated phases), stress power associated with each constituent should be accounted. Dissolution caused by external loading can also result in apparent viscoelastic behavior of cementitious materials. Predicting of cement paste viscoelasticity associated with stress power is more challengeable, and it highly depends on the accuracy of the thermodynamic properties utilized in the computations.

REFERENCES

[1] Haecker, C.J., E.J. Garboczi, J.W. Bullard, R.B. Bohn, Z. Sun, S.P. Shah, and T. Voigt, Modeling the linear elastic properties of Portland cement paste. Cement and Concrete Research, 2005. 35(10): p. 1948-1960.

[2] Pierre-Claude, A., Cements of yesterday and today: Concrete of tomorrow. Cement and Concrete Research, 2000. 30(9): p. 1349-1359.

[3] Grasley, Z.C., CAREER: Linking nanoscale and macroscale viscoelastic responses of cementitious materials. 2008, National Science Foundation.

[4] Wineman, A.S. and K.R. Rajagopal, Mechanical Response of Polymers: An Introduction. 2000, Cambridge, United Kingdom: Cambridge University Press

[5] Mindess, S., J.F. Young, and D. Darwin, Concrete. 2nd ed. 2002, Upper Saddle River, NJ: Prentice Hall.

[6] Jones, C.A., Contact Mechanics Based Mechanical Characterization of Portland Cement Paste, in Civil Engineering. 2011, Texas A&M University: Texas College Station. p. 183.

[7] Tamtsia, B.T. and J.J. Beaudoin, Basic creep of hardened cement paste A re-examination of the role of water. Cement and Concrete Research, 2000. 30(9): p. 1465-1475.

[8] Powers, T.C., Mechanisms of shrinkage and reversible creep of hardened portland cement paste Proceedings of International Conference On the Structure of Concrete. 1968, London, England: Cement and Concrete Association.

[9] Feldman, R.F. and P.J. Sereda, A model for hydrated portland cement paste as deduced from sorption-length change and mechanical properties. Materials and Structures, 1968. 1(6): p. 509-520.

[10] Ruetz, W., A hypothesis for the creep of hardened cement paste and the influence of simultaneous shrinkage Proceedings of International Conference On the Structure of Concrete. 1968, London, England: Cement and Concrete Association.

[11] Suter, M. and G. Benipal, Constitutive model for aging thermoviscoelasticity of reacting concrete I: theoretical formulation. Mechanics of Time-Dependent Materials, 2010. 14(3): p. 277-290.

[12] Bazant, Z.P., A.B. Hauggard, S. Baweja, and F.J. Ulm, Microprestress–solidification theory for concrete creep: I. Aging and drying effects. Journal of Engineering Mechanics, ASCE, 1997. 123(11): p. 1188-1194.

[13] Bentur, A., R.L. Berger, F.V. Lawrence, N.B. Milestone, S. Mindess, and J.F. Young, Creep and drying shrinkage of calcium silicates pastes: III. A hypothesis of irreversible strains. Cement and Concrete Research, 1979. 9(1): p. 83-95.

[14] Bazant, Z.P. and S. Prasannan, Solidification theory for Concrete creep: I. Formulation. Journal of Engineering Mechanics, ASCE, 1989. 115(8): p. 1691-1703.

[15] Carol, I. and Z.P. Bazant, Viscoelasticity with aging caused by solidification of nonaging constituent. Journal of Engineering Mechanics, ASCE, 1993. 119(11): p. 2252-2269.

[16] Grasley, Z.C., Measuring and Modeling the Time-Dependent Response of Cementitious Materials to Internal Stresses, in Civil Engineering. 2006, University of Illinois at Urbana-Champaign: Urbana, Illinois.

[17] Bazent, Z.P., Thermodynamics of solidifying or melting viscoelastic materials. Journal of Engineering Mechanics Division, ASCE, 1979. 105(6): p. 933-952.

[18] Grasley, Z.C., Closed-Form Solutions for Uniaxial Passive Restraint Experiments. American Concrete Institute, 2010. 270: p. 17-32.

[19] Becker, E.B., G.F. Carey, and J.T. Oden, Finite elment, An introduction, Volume 1. 1981, United States of America: Texas Institute for Computional Mechanics, The university of Texas at Austin.

[20] Garboczi, E.J., Finite Element and Finite Dierence Programs for Computing the Linear Electric and Elastic Properties of Digital Images of Random Materials. 1998, Building and Fire Research Laboratory,National Institute of Standards and Technology: Gaithersburg, Marylan.

[21] Bohn, R.B. and E.J. Garboczi, User Manual for Finite Element and Finite Difference Programs: A Parallel Version of NISTIR-6269. 2003, Gaithersburg: U.S. Department of commerce, Technology Administration, National Institute of Standards and Technology, Information Technology Laboratory, Building and Fire Research Labortory.

[22] Garboczi, E.J. and A.R. Day, An algorithm for computing the effective linear elastic properties of heterogeneous materials: Three-dimensional results for composites with equal phase poisson ratios. Journal of the Mechanics and Physics of Solids, 1995. 43(9): p. 1349-1362.

[23] Timnoshenko, S.P. and J.N. Goodier, Theory of Elasticity. 1970, New York,: McGraw-Hill.

[24] Landau, L.D. and E.M. Lifshitz, Theory of Elasticity. 3rd ed. 1986, Oxford: Pergamon.

[25] Bullard, J.W., B. Lothenbach, P.E. Stutzman, and K.A. Snyder, Coupling thermodynamics and digital image models to simulate hydration and microstructure development of portland cement pastes. Journal of Materials Research, 2011. 26(4): p. 609-622.

[26] Lothenbach, B. and F. Winnefeld, Thermodynamic modelling of the hydration of portland cement. Cement and Concrete Research, 2006. 36(2): p. 209-226.

[27] Haupt, R.K., Explanation of Final Report on Results of Tests for Portland Cement Proficiency Samples No.167 and No.168. 2008, Cement and Concrete Reference Laboratory: Gaithersburg, Maryland.

[28] Yang, T., B. Keller, and E. Magyari, AFM investigation of cement paste in humid air at different relative humidities. Journal of Physics D, 2002. 35(8): p. 25-28.

[29] Sáez de Ibarra, Y., J.J. Gaitero, E. Erkizia, and I. Campillo, Atomic force microscopy and nanoindentation of cement pastes with nanotube dispersions. Physica Status Solidi (A) Applications and Materials, 2006. 203(6): p. 1076-1081.

[30] Mondal, P., S.P. Shah, and L.D. Marks, Nanoscale Characterization of Cementitious Materials. American Concrete Institute, 2008. 105(2): p. 174-179.

[31] Grasley, Z.C., C.A. Jones, X. Li, E.J. Garboczi, and J.W. Bullard, Elastic and Viscoelastic Properties of Calcium Silicate Hydrate, in NICOM 4: 4th International Symposium on Nanotechnology in Consturction. 2012.

[32] Chen, J.J., L. Sorelli, M. Vandamme, F.-J. Ulm, and G. Chanvillard, A coupled nanoindentation/SEM-EDS study on low water/cement ratio portland cement paste: Evidence for C-S-H/$Ca(OH)_2$ nanocomposites. Journal of the American Ceramic Society, 2010. 93(5): p. 1484-1493.

[33] Trtik, P., B. Munch, and P. Lura, A critical examination of statistical nanoindentation on model materials and hardened cement pastes based on virtual experiments. Cement and Concrete Composites, 2009. 31(10): p. 705-714.

[34] Bower, A.F., Applied Mechanics of Solids. 2009, Brown University, Providence, Rhode Island, USA CRC Press; 1 edition

[35] Lubliner, J., Plasticity Theory (Revised Edition). 2008, University of California at Berkeley: Pearson Education, Inc.

[36] Fung, Y. and P. Tong, Classical and computational solid mechanics. Volume 1 of Advanced series in engineering science. 2001, Singapore: World Scientific Publishing Company.

[37] Vichit-Vadakan, W. and G.W. Scherer, Measuring permeability and stress relaxation of young cement paste by beam bending. Cement and Concrete Research, 2003. 33(12): p. 1925-1932.

[38] Pane, I. and W. Hansen, Early age creep and stress relaxation of concrete containing blended cements. Materials and Structures, 2002. 35(2): p. 92-96.

APPENDIX A

C++ CODE FOR THE DISSOLUTION-FORMATION VISCOELASTIC PROGRAM

This appendix is the C++ code used for simulating the aging viscoelastic behavior of cement paste. The code uses a main function and several sub functions that allow for the calculation of cement paste's viscoelastic moduli. Detailed introductions about these functions are shown in the code.

Before running the program, a series of input files each with a column of numbers should be prepared. Each number inside one file indicates the material phase in each voxel and each file represents the microstructure at a certain age. The names of the input files can be changed in the main function. Since this program is a strain controlled program, the controlled strain should also be input in the main function.

After running the program, one output file will be generated. This output file gives the six output stresses for the whole microstructure at each time step. The format of the output file should be:

strxx	stryy	strzz	strxz	stryz	strxy
3.9981	3.96747	4.0323	-0.0346229	-0.0139738	-0.0181493
strxx	stryy	strzz	strxz	stryz	strxy
3.9981	3.96747	4.0323	-0.0346229	-0.0139738	-0.0181493

…

Each line of number represents the output stress at each time step. Knowing the input controlled strain, the viscoelastic moduli can be calculated. The output content can also be modified in the main function together with the name of the output file.

Following is the code:

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <string>

using namespace std;

 /*
** ********************************************************
** BACKGROUND
*/

/*
** This program solves the linear equations in a random linear
** aging viscoelastic material, subject to an applied macroscopic strain,
** using the finite element method.  Each pixel in the 3-D digital
** image is a cubic tri-linear finite element,  having its own
** moduli tensor. Periodic boundary conditions are maintained.
** In the comments below, (USER) means that this is a section of code that
** the user might have to change for his particular problem. Therefore the
** user is encouraged to search for this string.
*/

/*
** PROBLEM AND VARIABLE DEFINITION
*/

/*
** The problem being solved is the minimization of the energy
** 1/2 uAu + b u + C, where A is the history dependent matrix composed of the
** stiffness matrices (dk) for each pixel/element, b is a vector
** and C is a constant that are determined by the applied strain,the periodic boundary
** conditions and the aging status of each pixel, and u is a vector of
** all the displacements. The solution
```

```
   ** method used is the conjugate gradient relaxation algorithm.
   ** Other variables are:  gb is the gradient = Au+b, h and Ah are
   ** auxiliary variables used in the conjugate gradient algorithm (in dembx),
   ** dk(n,i,j) is the stiffness matrix of the n'th phase, cmod(n,i,j) is
   ** the elastic moduli tensor of the n'th phase, pix is a vector that gives
   ** the phase label of each pixel, ib is a matrix that gives the labels of
   ** the 27 (counting itself) neighbors of a given node, prob is the volume
   ** fractions of the various phases,
   ** strxx, stryy, strzz, strxz, stryz, and strxy are the six Voigt
   ** volume averaged total stresses, and
   ** sxx, syy, szz, sxz, syz, and sxy are the six Voigt
   ** volume averaged total strains.
   */

   /*
   ** DIMENSIONS
   */

   /*
   ** The vectors u,gb,b,h, and Ah are dimensioned to be the system size,
   ** ns=nx*ny*nz, with three components, where the digital image of the
   ** microstructure considered is a rectangular paralleliped, nx x ny x nz
   ** in size.  The arrays pix and ib are are also dimensioned to the system size.
   ** The array ib has 27 components, for the 27 neighbors of a node.
   ** Note that the program is set up at present to have at most 100
   ** different phases.  This can easily be changed, simply by changing
   ** the dimensions of dk, prob, and cmod. The parameter nphase gives the
   ** number of phases being considered in the problem.
   ** All arrays are passed between subroutines using simple common statements.
   */

   /*
   ** (USER) Change these dimensions and in other subroutines at same time.
   */



/*
**Global variables.
*/
double u[1000001][51][4],gb[1000001][4],b[1000001][4];
double h[1000001][4],Ah[1000001][4];
double cmod[101][7][7][51],dk[101][9][4][9][4][51];
```

```
long ib[1000001][28];
short pix[1000001],pixstor[1000001],pixt[1000001];
int phasechange[1000001];

double strxx, stryy, strzz, strxz, stryz, strxy, sxx, syy, szz, sxz, syz, sxy;
double exx, eyy, ezz, exz, eyz, exy, C;

double str11, str22, str33, str13, str23, str12, s11, s22, s33, s13, s23, s12;
double dxx, dyy,dzz, dxy, dxz, dyz;

/*
**Subfuction that sets up microstructural image.
*/

void ppixel(int nx, int ny, int nz, int ns, int nphase, char filename[15])
{
   ifstream fin(filename);
   ofstream fout("outputfile.txt");

   int nxy=nx*ny;
   for(int k=1; k<=nz; k++)
   {
      for (int j=1; j<=ny; j++)
      {
         for (int i=1; i<=nx; i++)
         {
            int m=nxy*(k-1)+nx*(j-1)+i;
            fin>>pix[m];
         }
      }
   }

   /*
   **(USER)Orgnize the phases and combine the phases with the same moduli for
   **memory saving purpose.
   */
   for (int m = 1; m <= ns; m++ )
   {
      if(pix[m]==1)
      {
         pix[m]=1;
      }
      else if(pix[m]==2||pix[m]==3||pix[m]==4||pix[m]==5)
      {
```

```
            pix[m]=2;
        }
        else if(pix[m]==6||pix[m]==7||pix[m]==11||pix[m]==13||pix[m]==14||pix[m]==17||
                pix[m]==18)
        {
            pix[m]=3;
        }
        else if(pix[m]==8)
        {
            pix[m]=4;
        }
        else if(pix[m]==9)
        {
            pix[m]=5;
        }
        else if(pix[m]==10)
        {
            pix[m]=6;
        }
        else if(pix[m]==12)
        {
            pix[m]=7;
        }
        else if(pix[m]==15||pix[m]==16)
        {
            pix[m]=8;
        }

        /*
        **Check for wrong phase labels--less than 1 or greater than nphase.
        */
        if(pix[m]<1)
        {
            fout<< "Phase label in pix < 1--error at "<<m<<"\n";
        }

        if(pix[m]>nphase)
        {
            fout<< "Phase label in pix > nphase--error at "<<m<<"\n";
        }
    }
}

/*
```

```
**Subfuction that counts volume fractions. This fuction is not required in this program
**but it is for good reference.
*/
void  assig(int ns,int nphase,double (&prob)[101])
{
   for(int i=1; i<=nphase; i++)
   {
      prob[i]=0.0;
   }

   for (int m=1; m<=ns; m++)
   {
      for (i=1; i<=nphase; i++)
      {
         if(pix[m]==i)
         {
            prob[i]=prob[i]+1;
         }
      }
   }

   for (i=1; i<=nphase; i++)
   {
      prob[i]=prob[i]/(ns*1.0);
   }
}

/*
**Subfuction that caculates the strain in one pixel at time qt.
*/
void strain (int i,int j,int  k,int nx,int ny,int  nz,int qt)
{
   double  dndx[9],dndy[9],dndz[9],es[7][9][4],uu[9][4];
   int nxy=nx*ny;

   /*
   ** set up single element strain matrix
   ** dndx, dndy, and dndz are the components of the average strain
   ** matrix in a pixel
   */

   dndx[1]=-0.25;
   dndx[2]=0.25;
   dndx[3]=0.25;
```

```
dndx[4]=-0.25;
dndx[5]=-0.25;
dndx[6]=0.25;
dndx[7]=0.25;
dndx[8]=-0.25;
dndy[1]=-0.25;
dndy[2]=-0.25;
dndy[3]=0.25;
dndy[4]=0.25;
dndy[5]=-0.25;
dndy[6]=-0.25;
dndy[7]=0.25;
dndy[8]=0.25;
dndz[1]=-0.25;
dndz[2]=-0.25;
dndz[3]=-0.25;
dndz[4]=-0.25;
dndz[5]=0.25;
dndz[6]=0.25;
dndz[7]=0.25;
dndz[8]=0.25;

/*
** Build averaged strain matrix, follows code in femat, but for average
** strain over the pixel, not the strain at a point.
*/
for (int n1 = 1; n1 <= 6; n1++ )
{
   for (int n2 = 1; n2 <= 8; n2++ )
   {
      for (int n3 = 1; n3 <= 3; n3++ )
      {
         es[n1][n2][n3]=0.0;
      }
   }
}

for (int n = 1; n <= 8; n++ )
{
   es[1][n][1]=dndx[n];
   es[2][n][2]=dndy[n];
   es[3][n][3]=dndz[n];
   es[4][n][1]=dndz[n];
   es[4][n][3]=dndx[n];
```

```
   es[5][n][2]=dndz[n];
   es[5][n][3]=dndy[n];
   es[6][n][1]=dndy[n];
   es[6][n][2]=dndx[n];
}

dxx=0.0;
dyy=0.0;
dzz=0.0;
dxz=0.0;
dyz=0.0;
dxy=0.0;

int m1=( k-1 )*nxy+( j-1 )*nx+i;
for (int mm =1; mm <= 3; mm++ )
{
   uu[1][mm]=u[m1][qt][mm];
   uu[2][mm]=u[ib[m1][3]][qt][mm];
   uu[3][mm]=u[ib[m1][2]][qt][mm];
   uu[4][mm]=u[ib[m1][1]][qt][mm];
   uu[5][mm]=u[ib[m1][26]][qt][mm];
   uu[6][mm]=u[ib[m1][19]][qt][mm];
   uu[7][mm]=u[ib[m1][18]][qt][mm];
   uu[8][mm]=u[ib[m1][17]][qt][mm];
}

if( i==nx )
{
   uu[2][1]=uu[2][1]+exx*nx;
   uu[2][2]=uu[2][2]+exy*nx;
   uu[2][3]=uu[2][3]+exz*nx;
   uu[3][1]=uu[3][1]+exx*nx;
   uu[3][2]=uu[3][2]+exy*nx;
   uu[3][3]=uu[3][3]+exz*nx;
   uu[6][1]=uu[6][1]+exx*nx;
   uu[6][2]=uu[6][2]+exy*nx;
   uu[6][3]=uu[6][3]+exz*nx;
   uu[7][1]=uu[7][1]+exx*nx;
   uu[7][2]=uu[7][2]+exy*nx;
   uu[7][3]=uu[7][3]+exz*nx;
}
if( j==ny )
{
   uu[3][1]=uu[3][1]+exy*ny;
```

```
    uu[3][2]=uu[3][2]+eyy*ny;
    uu[3][3]=uu[3][3]+eyz*ny;
    uu[4][1]=uu[4][1]+exy*ny;
    uu[4][2]=uu[4][2]+eyy*ny;
    uu[4][3]=uu[4][3]+eyz*ny;
    uu[7][1]=uu[7][1]+exy*ny;
    uu[7][2]=uu[7][2]+eyy*ny;
    uu[7][3]=uu[7][3]+eyz*ny;
    uu[8][1]=uu[8][1]+exy*ny;
    uu[8][2]=uu[8][2]+eyy*ny;
    uu[8][3]=uu[8][3]+eyz*ny;
}
if( k==nz )
{
    uu[5][1]=uu[5][1]+exz*nz;
    uu[5][2]=uu[5][2]+eyz*nz;
    uu[5][3]=uu[5][3]+ezz*nz;
    uu[6][1]=uu[6][1]+exz*nz;
    uu[6][2]=uu[6][2]+eyz*nz;
    uu[6][3]=uu[6][3]+ezz*nz;
    uu[7][1]=uu[7][1]+exz*nz;
    uu[7][2]=uu[7][2]+eyz*nz;
    uu[7][3]=uu[7][3]+ezz*nz;
    uu[8][1]=uu[8][1]+exz*nz;
    uu[8][2]=uu[8][2]+eyz*nz;
    uu[8][3]=uu[8][3]+ezz*nz;
}

/*
** Local strains in a pixel.
*/
for (int n3 =1; n3 <= 3; n3++ )
{
    for (int n8 =1; n8 <= 8; n8++ )
    {
        dxx=dxx+es[1][n8][n3]
            *uu[n8][n3];
        dyy=dyy+es[2][n8][n3]
            *uu[n8][n3];
        dzz=dzz+es[3][n8][n3]
            *uu[n8][n3];
        dxz=dxz+es[4][n8][n3]
            *uu[n8][n3];
        dyz=dyz+es[5][n8][n3]
```

```
            *uu[n8][n3];
          dxy=dxy+es[6][n8][n3]
            *uu[n8][n3];
      }
   }
}

/*
** Subfuction that sets up the moduli variables,
** the stiffness matrices,dk, the linear term in
** displacements, b, and the constant term, C, that appear in the total energy
** due to the periodic boundary conditions.
*/
void femat (int nx, int ny,int nz,int  ns,double (&phasemod)[101][51][3],int  nphase,
             int q )
{
   double  dndx[9],dndy[9],dndz[9];
   double  g[4][4][4],ck[7][7],cmu[7][7];
   double  es[7][9][4],delta[9][4];
   int is[9];

   int nxy=nx*ny;

   /*
   **An anisotropic matrix could be directly input at any point,and  program is written
   ** to be set as a general elastic moduli tensor at each time step, but is only explicitly
   ** implemented for isotropic materials.
   */

   /*
   ** initialize stiffness matrices
   */
   for (int m = 1; m <= nphase; m++ )
   {
      for (int l = 1; l <=3; l++ )
      {
         for (int k = 1; k <= 3; k++ )
         {
            for ( int j = 1; j <= 8; j++ )
            {
               for (int i = 1; i <=8; i++ )
               {
                  dk[m][i][k][j][l][q]=0.0;
               }
```

```
            }
        }
    }
}

/*
**Set up elastic moduli matrices for each kind of element
** ck and cmu are the Bulk and Shear modulus matrices, which need to be
** weighted by the actual Bulk and Shear moduli.
*/
ck[1][1]=1.0;
ck[1][2]=1.0;
ck[1][3]=1.0;
ck[1][4]=0.0;
ck[1][5]=0.0;
ck[1][6]=0.0;
ck[2][1]=1.0;
ck[2][2]=1.0;
ck[2][3]=1.0;
ck[2][4]=0.0;
ck[2][5]=0.0;
ck[2][6]=0.0;
ck[3][1]=1.0;
ck[3][2]=1.0;
ck[3][3]=1.0;
ck[3][4]=0.0;
ck[3][5]=0.0;
ck[3][6]=0.0;
ck[4][1]=0.0;
ck[4][2]=0.0;
ck[4][3]=0.0;
ck[4][4]=0.0;
ck[4][5]=0.0;
ck[4][6]=0.0;
ck[5][1]=0.0;
ck[5][2]=0.0;
ck[5][3]=0.0;
ck[5][4]=0.0;
ck[5][5]=0.0;
ck[5][6]=0.0;
ck[6][1]=0.0;
ck[6][2]=0.0;
ck[6][3]=0.0;
ck[6][4]=0.0;
```

```
ck[6][5]=0.0;
ck[6][6]=0.0;

cmu[1][1]=4.0/3.0;
cmu [1][2]=-2.0/3.0;
cmu[1][3]=-2.0/3.0;
cmu[1][4]=0.0;
cmu[1][5]=0.0;
cmu[1][6]=0.0;
cmu[2][1]=-2.0/3.0;
cmu[2][2]=4.0/3.0;
cmu[2][3]=-2.0/3.0;
cmu[2][4]=0.0;
cmu[2][5]=0.0;
cmu[2][6]=0.0;
cmu[3][1]=-2.0/3.0;
cmu[3][2]=-2.0/3.0;
cmu[3][3]=4.0/3.0;
cmu[3][4]=0.0;
cmu[3][5]=0.0;
cmu[3][6]=0.0;
cmu[4][1]=0.0;
cmu[4][2]=0.0;
cmu[4][3]=0.0;
cmu[4][4]=1.0;
cmu[4][5]=0.0;
cmu[4][6]=0.0;
cmu[5][1]=0.0;
cmu[5][2]=0.0;
cmu[5][3]=0.0;
cmu[5][4]=0.0;
cmu[5][5]=1.0;
cmu[5][6]=0.0;
cmu[6][1]=0.0;
cmu[6][2]=0.0;
cmu[6][3]=0.0;
cmu[6][4]=0.0;
cmu[6][5]=0.0;
cmu[6][6]=1.0;

for (int k = 1; k <= nphase; k++ )
{
   for (int j = 1; j <= 6; j++ )
   {
```

```
      for (int i = 1; i <= 6; i++ )
      {
            cmod[k][i][j][q]=phasemod[k][q][1]*ck[i][j]+
                              phasemod[k][q][2]*cmu[i][j];
      }
   }
}

/*
** Set up Simpson's integration rule weight vector.
*/
for ( k = 1; k <= 3; k++ )
{
   for (int j = 1; j <= 3; j++ )
   {
      for (int i = 1; i <= 3; i++ )
      {
         int   nm=0;
         if( i==2 )
         {
            nm=nm+1;
         }
         if( j==2 )
         {
            nm=nm+1;
         }
         if( k==2 )
         {
            nm=nm+1;
         }
         g[i][j][k]=pow(4.0,nm);
      }
   }
}

/*
** Loop over the nphase kinds of pixels and Simpson's rule quadrature
** points in order to compute the stiffness matrices.  Stiffness matrices
** of trilinear finite elements are quadratic in x, y, and z, so that
** Simpson's rule quadrature gives exact results.
*/
for ( int  ijk = 1; ijk <= nphase; ijk++ )
{
   for ( k = 1; k <= 3; k++ )
```

```
{
   for (int j = 1; j <= 3; j++ )
   {
      for (int i = 1; i <= 3; i++ )
      {
         double     x=1.0*( i-1 )/2.0;
         double     y=1.0*( j-1 )/2.0;
         double   z=1.0*( k-1 )/2.0;
         /*
         ** dndx means the negative derivative, with respect to x, of the shape
         ** matrix, dndy, and dndz are similar.
         */
         dndx[1]=-( 1.0-y )*( 1.0-z );
         dndx[2]=( 1.0-y )*( 1.0-z );
         dndx[3]=y*( 1.0-z );
         dndx[4]=-y*( 1.0-z );
         dndx[5]=-( 1.0-y )*z;
         dndx[6]=( 1.0-y )*z;
         dndx[7]=y*z;
         dndx[8]=-y*z;
         dndy[1]=-( 1.0-x )*( 1.0-z );
         dndy[2]=-x*( 1.0-z );
         dndy[3]=x*( 1.0-z );
         dndy[4]=( 1.0-x )*( 1.0-z );
         dndy[5]=-( 1.0-x )*z;
         dndy[6]=-x*z;
         dndy[7]=x*z;
         dndy[8]=( 1.0-x )*z;
         dndz[1]=-( 1.0-x )*( 1.0-y );
         dndz[2]=-x*( 1.0-y );
         dndz[3]=-x*y;
         dndz[4]=-( 1.0-x )*y;
         dndz[5]=( 1.0-x )*( 1.0-y );
         dndz[6]=x*( 1.0-y );
         dndz[7]=x*y;
         dndz[8]=( 1.0-x )*y;

         /*
         ** Build strain matrix.
         */
         for ( int n1 = 1; n1 <= 6; n1++ )
         {
            for (int  n2 = 1; n2 <= 8; n2++ )
            {
```

```
      for (int  n3 = 1; n3 <= 3; n3++ )
      {
      es[n1][n2][n3]=0.0;
      }
   }
}
for (int  n = 1; n <= 8; n++ )
{
   es[1][n][1]=dndx[n];
   es[2][n][2]=dndy[n];
   es[3][n][3]=dndz[n];
   es[4][n][1]=dndz[n];
   es[4][n][3]=dndx[n];
   es[5][n][2]=dndz[n];
   es[5][n][3]=dndy[n];
   es[6][n][1]=dndy[n];
   es[6][n][2]=dndx[n];

}
/*
** Matrix multiply to determine value at (x,y,z), multiply by
** proper weight, and sum into dk, the stiffness matrix.
*/
for (int  mm = 1; mm <= 3; mm++ )
{
   for ( int nn = 1; nn <= 3; nn++ )
   {
      for ( int ii = 1; ii <= 8; ii++ )
      {
         for (int jj = 1; jj <= 8; jj++ )
         {
            /*
            ** Define sum over strain matrices and elastic moduli matrix for
            ** stiffness matrix.
            */
            double sum=0.0;
            for (int kk = 1; kk <= 6; kk++ )
            {
               for (int ll = 1; ll <= 6; ll++ )
               {
                  sum=sum+es[kk][ii][mm]*cmod[ijk][kk][ll][q]
                      *es[ll][jj][nn];
               }
            }
```

```
                        dk[ijk][ii][mm][jj][nn][q]=dk[ijk][ii][mm][jj][nn][q]
                                +g[i][j][k]*sum/216.;
                    }
                }
            }
        }
    }
}
}


/*
** Set up vector for term, b, and constant term, C,
** in the energy.  This is done using the stiffness matrices,
** and the periodic terms in the applied strain that come in at the
** boundary pixels via the periodic boundary conditions and the
** condition that an applied macroscopic strain exists.
** It is easier to set b up this way than to analytically
** write out all the terms involved.
*/

/*
** Initialize b and C.
*/
int qtime;

for (int  m3 = 1; m3 <= 3; m3++ )
{
   for ( m = 1; m <= ns; m++ )
   {
      b[m][m3]=0.0;
   }
}
C=0.0;

/*
** For all cases, the correspondence between 1-8 finite element node
** labels and 1-27 neighbor labels is :
** 1:ib(m,27), 2:ib(m,3),
** 3:ib(m,2),4:ib(m,1),
** 5:ib(m,26),6:ib(m,19)
** 7:ib(m,18),8:ib(m,17)
*/
```

```
is[1]=27;
is[2]=3;
is[3]=2;
is[4]=1;
is[5]=26;
is[6]=19;
is[7]=18;
is[8]=17;

/*
**For aging, the b matrix and the value of C should be given a value
**counting the existence of free strain. If aging occurs in one pixel,
**calculate the free strainin that pixel and deduct the value of free strain.
*/
for ( k = 1; k <= nz; k++ )
{
   for (int j = 1; j <= ny; j++ )
   {
      for (int i = 1; i <= nx; i++ )
      {
         m=nxy*( k-1 )+(j-1)*nx+i;
         /*
         **if aging occurs
         */
         if (phasechange[m]==1)
         {
            /*
            ** Calculate the strain in that pixel at aged time.
            */
            strain(i,j,k,nx,ny,nz,pixt[m]);
            /*
            ** Correct b and C value for aged pixels.
            */
            for (int i8 = 1; i8 <= 8; i8++ )
            {
               for ( int i3 = 1; i3 <= 3; i3++ )
               {
                  delta[i8][i3]=0.0;
               }

               if( i8==2 )
               {
                  delta[i8][1]=delta[i8][1]-dxx;
                  delta[i8][2]=delta[i8][2]-dxy;
```

```
      delta[i8][3]=delta[i8][3]-dxz;
   }
   if( i8==4 )
   {
      delta[i8][1]=delta[i8][1]-dxy;
      delta[i8][2]=delta[i8][2]-dyy;
      delta[i8][3]=delta[i8][3]-dyz;
   }
   if( i8==5 )
   {
      delta[i8][1]=delta[i8][1]-dxz;
      delta[i8][2]=delta[i8][2]-dyz;
      delta[i8][3]=delta[i8][3]-dzz;
   }
   if( i8==8 )
   {
      delta[i8][1]=delta[i8][1]-dxy-dxz;
      delta[i8][2]=delta[i8][2]-dyy-dyz;
      delta[i8][3]=delta[i8][3]-dyz-dzz;
   }
   if( i8==6 )
   {
      delta[i8][1]=delta[i8][1]-dxx-dxz;
      delta[i8][2]=delta[i8][2]-dxy-dyz;
      delta[i8][3]=delta[i8][3]-dxz-dzz;
   }
   if( i8==3 )
   {
      delta[i8][1]=delta[i8][1]-dxx-dxy;
      delta[i8][2]=delta[i8][2]-dxy-dyy;
      delta[i8][3]=delta[i8][3]-dxz-dyz;
   }
   if( i8==7 )
   {
      delta[i8][1]=delta[i8][1]-dxx-dxy-dxz;
      delta[i8][2]=delta[i8][2]-dxy-dyy-dyz;
      delta[i8][3]=delta[i8][3]-dxz-dyz-dzz;
   }
}

for (int nn = 1; nn <= 3; nn++ )
{
   for (int mm = 1; mm <= 8; mm++ )
   {
```

```
                    double sum=0.0;
                    for ( m3 = 1; m3 <= 3; m3++ )
                    {
                       for (int m8 = 1; m8 <= 8; m8++ )
                       {
                          for (qtime=pixt[m]+1; qtime <=pixt[m]+1; qtime++)
                          {
                          sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn]
                                [qtime-pixt[m]];

                          C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                                [nn][qtime-pixt[m]] *delta[mm][nn];
                          }
                       }
                    }
                    b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
                 }
              }
           }
        }
     }
}

/*
**For viscoelastic phases, the b matrix and the value of C  should be given a value
** counting the existence of free strain. If aging occurs in one pixel,
**calculate the free strain in that pixel and deduct the value of free strain.
*/
for ( k = 1; k <= nz; k++ )
{
   for (int j = 1; j <= ny; j++ )
   {
      for (int i = 1; i <= nx; i++ )
      {
         m=nxy*( k-1 )+(j-1)*nx+i;

         /*
         ** If the pixel is viscoelastic
         */
         if (pix[m]==7)
         {
            /*
            **Calculate the strain in these pixels for all previous steps and adds them **
            up.
```

```
*/
for (qtime=pixt[m]+1; qtime <=q-1; qtime++)
{
   strain(i,j,k,nx,ny,nz,qtime);

   for (int i8 = 1; i8 <= 8; i8++ )
   {
      for ( int i3 = 1; i3 <= 3; i3++ )
      {
         delta[i8][i3]=0.0;
      }

      if( i8==2 )
      {
         delta[i8][1]=delta[i8][1]-dxx;
         delta[i8][2]=delta[i8][2]-dxy;
         delta[i8][3]=delta[i8][3]-dxz;
      }
      if( i8==4 )
      {
         delta[i8][1]=delta[i8][1]-dxy;
         delta[i8][2]=delta[i8][2]-dyy;
         delta[i8][3]=delta[i8][3]-dyz;
      }
      if( i8==5 )
      {
         delta[i8][1]=delta[i8][1]-dxz;
         delta[i8][2]=delta[i8][2]-dyz;
         delta[i8][3]=delta[i8][3]-dzz;
      }
      if( i8==8 )
      {
         delta[i8][1]=delta[i8][1]-dxy-dxz;
         delta[i8][2]=delta[i8][2]-dyy-dyz;
         delta[i8][3]=delta[i8][3]-dyz-dzz;
      }
      if( i8==6 )
      {
         delta[i8][1]=delta[i8][1]-dxx-dxz;
         delta[i8][2]=delta[i8][2]-dxy-dyz;
         delta[i8][3]=delta[i8][3]-dxz-dzz;
      }
      if( i8==3 )
      {
```

```
            delta[i8][1]=delta[i8][1]-dxx-dxy;
            delta[i8][2]=delta[i8][2]-dxy-dyy;
            delta[i8][3]=delta[i8][3]-dxz-dyz;
         }
         if( i8==7 )
         {
            delta[i8][1]=delta[i8][1]-dxx-dxy-dxz;
            delta[i8][2]=delta[i8][2]-dxy-dyy-dyz;
            delta[i8][3]=delta[i8][3]-dxz-dyz-dzz;
         }
      }
      for (int nn = 1; nn <= 3; nn++ )
      {
         for (int mm = 1; mm <= 8; mm++ )
         {
            double sum=0.0;
            for ( m3 = 1; m3 <= 3; m3++ )
            {
               for (int m8 = 1; m8 <= 8; m8++ )
               {
                  sum=sum-delta[m8][m3]*
                        dk[pix[m]][m8][m3][mm][nn][q-qtime+1];

                  C=C+0.5-delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                        [nn][q-qtime+1]*delta[mm][nn];
               }
            }
          b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
         }
      }
   }

   /*
   **If aging has occured in the viscoelastic pixel, the strain in the pixel when
   **aging occurs should be deducted in all time steps after aging occures.
   */
   if(phasechange[m]==1)
   {
      strain(i,j,k,nx,ny,nz,pixt[m]);

      for (int i8 = 1; i8 <= 8; i8++ )
      {
         for ( int i3 = 1; i3 <= 3; i3++ )
         {
```

```
   delta[i8][i3]=0.0;
}

if( i8==2 )
{
   delta[i8][1]=delta[i8][1]-dxx;
   delta[i8][2]=delta[i8][2]-dxy;
   delta[i8][3]=delta[i8][3]-dxz;
}
if( i8==4 )
{
   delta[i8][1]=delta[i8][1]-dxy;
   delta[i8][2]=delta[i8][2]-dyy;
   delta[i8][3]=delta[i8][3]-dyz;
}
if( i8==5 )
{
   delta[i8][1]=delta[i8][1]-dxz;
   delta[i8][2]=delta[i8][2]-dyz;
   delta[i8][3]=delta[i8][3]-dzz;
}
if( i8==8 )
{
   delta[i8][1]=delta[i8][1]-dxy-dxz;
   delta[i8][2]=delta[i8][2]-dyy-dyz;
   delta[i8][3]=delta[i8][3]-dyz-dzz;
}
if( i8==6 )
{
   delta[i8][1]=delta[i8][1]-dxx-dxz;
   delta[i8][2]=delta[i8][2]-dxy-dyz;
   delta[i8][3]=delta[i8][3]-dxz-dzz;
}
if( i8==3 )
{
   delta[i8][1]=delta[i8][1]-dxx-dxy;
   delta[i8][2]=delta[i8][2]-dxy-dyy;
   delta[i8][3]=delta[i8][3]-dxz-dyz;
}
if( i8==7 )
{
   delta[i8][1]=delta[i8][1]-dxx-dxy-dxz;
   delta[i8][2]=delta[i8][2]-dxy-dyy-dyz;
   delta[i8][3]=delta[i8][3]-dxz-dyz-dzz;
```

```
                }
            }

            for (int nn = 1; nn <= 3; nn++ )
            {
               for (int mm = 1; mm <= 8; mm++ )
               {
                  double sum=0.0;
                  for ( m3 = 1; m3 <= 3; m3++ )
                  {
                     for (int m8 = 1; m8 <= 8; m8++ )
                     {
                        for (qtime=2; qtime <=q-pixt[m]; qtime++)
                        {
                           sum=sum+delta[m8][m3]*dk[pix[m]]
                              [m8][m3][mm][nn][qtime];

                           C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                              [nn][qtime]*delta[mm][nn];
                        }
                     }
                  }
                b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
               }
            }
         }
      }
   }
}

/*
**For boundary conditions
*/

/*
** x=nx face
*/
for ( int i3 = 1; i3 <= 3; i3++ )
{
   for (int i8 = 1; i8 <= 8; i8++ )
   {
      delta[i8][i3]=0.0;
      if( i8==2||i8==3||i8==6||i8==7 )
```

```
        {
          delta[i8][1]=exx*nx;
          delta[i8][2]=exy*nx;
          delta[i8][3]=exz*nx;
        }
      }
    }
    for (int j = 1; j <= ny-1; j++ )
    {
      for ( k = 1; k <= nz-1; k++ )
      {
        m=nxy*( k-1 )+j*nx;
        for (int nn = 1; nn <= 3; nn++ )
        {
          for (int mm = 1; mm <= 8; mm++ )
          {
            double sum=0.0;
            for ( m3 = 1; m3 <= 3; m3++ )
            {
              for (int m8 = 1; m8 <= 8; m8++ )
              {
                if (phasechange[m]==1)
                {
                  for (qtime=pixt[m]+1; qtime <=q; qtime++)
                  {
                    sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                          [nn][qtime-pixt[m]];

                    C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                          [nn][qtime-pixt[m]]*delta[mm][nn];
                  }
                }

                if (phasechange[m]==0)
                {
                  for (qtime=1; qtime <=q; qtime++)
                  {
                    sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime];
                    C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime]
                        *delta[mm][nn];
                  }
                }
              }
            }
```

```
            b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
         }
      }
   }
}

/*
** y=ny face
*/
for ( i3 = 1; i3 <= 3; i3++ )
{
   for (int i8 = 1; i8 <= 8; i8++ )
   {
      delta[i8][i3]=0.0;
      if( i8==3||i8==4||i8==7||i8==8 )
      {
         delta[i8][1]=exy*ny;
         delta[i8][2]=eyy*ny;
         delta[i8][3]=eyz*ny;
      }
   }
}
for (int i = 1; i <= nx-1; i++ )
{
   for ( k = 1; k <= nz-1; k++ )
   {
      m=nxy*( k-1 )+nx*( ny-1 )+i;
      for (int nn = 1; nn <= 3; nn++ )
      {
         for (int mm = 1; mm <= 8; mm++ )
         {
            double sum=0.0;
            for ( m3 = 1; m3 <= 3; m3++ )
            {
               for (int m8 = 1; m8 <= 8; m8++ )
               {
                  if (phasechange[m]==1)
                  {
                     for (qtime=pixt[m]+1; qtime <=q; qtime++)
                     {
                        sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3]
                              [mm][nn][qtime-pixt[m]];

                        C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3]
```

```
                        [mm][nn][qtime-pixt[m]] *delta[mm][nn];
                    }
                }

                if (phasechange[m]==0)
                {
                    for (qtime=1; qtime <=q; qtime++)
                    {
                        sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime];

                        C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime]
                            *delta[mm][nn];
                    }
                }
            }
        }
        b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
    }
  }
 }
}

/*
** z=nz face
*/
for ( i3 = 1; i3 <= 3; i3++ )
{
   for (int i8 = 1; i8 <= 8; i8++ )
   {
      delta[i8][i3]=0.0;
      if( i8==5||i8==6||i8==7||i8==8 )
      {
         delta[i8][1]=exz*nz;
         delta[i8][2]=eyz*nz;
         delta[i8][3]=ezz*nz;
      }
   }
}
for ( i = 1; i <= nx-1; i++ )
{
   for ( j = 1; j <= ny-1; j++ )
   {
      m=nxy*( nz-1 )+nx*( j-1 )+i;
      for (int nn = 1; nn <= 3; nn++ )
```

```
        {
           for (int mm = 1; mm <= 8; mm++ )
           {
              double sum=0.0;
              for ( m3 = 1; m3 <= 3; m3++ )
              {
                 for (int m8 = 1; m8 <= 8; m8++ )
                 {
                    if (phasechange[m]==1)
                    {
                       for (qtime=pixt[m]+1; qtime <=q; qtime++)
                       {
                          sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3]
                                [mm][nn][qtime-pixt[m]];

                          C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                                [nn][qtime-pixt[m]]*delta[mm][nn];
                       }
                    }

                    if (phasechange[m]==0)
                    {
                       for (qtime=1; qtime <=q; qtime++)
                       {
                          sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime];

                          C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime]
                             *delta[mm][nn];
                       }
                    }
                 }
              }
              b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
           }
        }
     }
  }
/*
** x=nx y=ny edge
*/
for ( i3 = 1; i3 <= 3; i3++ )
{
   for (int i8 = 1; i8 <= 8; i8++ )
   {
```

```
        delta[i8][i3]=0.0;
        if( i8==2||i8==6 )
        {
           delta[i8][1]=exx*nx;
           delta[i8][2]=exy*nx;
           delta[i8][3]=exz*nx;
        }
        if( i8==4||i8==8 )
        {
           delta[i8][1]=exy*ny;
           delta[i8][2]=eyy*ny;
           delta[i8][3]=eyz*ny;
        }
        if( i8==3||i8==7 )
        {
           delta[i8][1]=exy*ny+exx*nx;
           delta[i8][2]=eyy*ny+exy*nx;
           delta[i8][3]=eyz*ny+exz*nx;
        }
     }
  }
  for ( k = 1; k <= nz-1; k++ )
  {
     m=nxy*k;
     for (int nn = 1; nn <= 3; nn++ )
     {
        for (int mm = 1; mm <= 8; mm++ )
        {
           double sum=0.0;
           for ( m3 = 1; m3 <= 3; m3++ )
           {
              for (int m8 = 1; m8 <= 8; m8++ )
              {
                 if (phasechange[m]==1)
                 {
                    for (qtime=pixt[m]+1; qtime <=q; qtime++)
                    {
                       sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3]
                          [mm][nn][qtime-pixt[m]];

                       C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                          [nn][qtime-pixt[m]]*delta[mm][nn];
                    }
                 }
```

```
                if (phasechange[m]==0)
                {
                   for (qtime=1; qtime <=q; qtime++)
                   {
                      sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime];

                      C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime]
                          *delta[mm][nn];
                   }
                }
             }
          }
          b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
       }
    }
}
/*
** x=nx z=nz edge
*/
for ( i3 = 1; i3 <= 3; i3++ )
{
   for (int i8 = 1; i8 <= 8; i8++ )
   {
      delta[i8][i3]=0.0;
      if( i8==2||i8==3 )
      {
         delta[i8][1]=exx*nx;
         delta[i8][2]=exy*nx;
         delta[i8][3]=exz*nx;
      }
      if( i8==5||i8==8 )
      {
         delta[i8][1]=exz*nz;
         delta[i8][2]=eyz*nz;
         delta[i8][3]=ezz*nz;
      }
      if( i8==6||i8==7 )
      {
         delta[i8][1]=exz*nz+exx*nx;
         delta[i8][2]=eyz*nz+exy*nx;
         delta[i8][3]=ezz*nz+exz*nx;
      }
   }
```

```
}
for ( j = 1; j <= ny-1; j++ )
{
   m=nxy*( nz-1 )+nx*( j-1 )+nx;
   for (int nn = 1; nn <= 3; nn++ )
   {
      for (int mm = 1; mm <= 8; mm++ )
      {
         double sum=0.0;
         for ( m3 = 1; m3 <= 3; m3++ )
         {
            for (int m8 = 0; m8 <= 8; m8++ )
            {
               if (phasechange[m]==1)
               {
                  for (qtime=pixt[m]+1; qtime <=q; qtime++)
                  {
                     sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3]
                        [mm][nn][qtime-pixt[m]];

                     C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                        [nn][qtime-pixt[m]]*delta[mm][nn];
                  }
               }

               if (phasechange[m]==0)
               {
                  for (qtime=1; qtime <=q; qtime++)
                  {
                     sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime];

                     C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime]
                       *delta[mm][nn];
                  }
               }
            }
         }
         b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
      }
   }
}
/*
** y=ny z=nz edge
*/
```

```
for ( i3 = 1; i3 <= 3; i3++ )
{
   for (int i8 = 1; i8 <= 8; i8++ )
   {
      delta[i8][i3]=0.0;
      if( i8==5||i8==6 )
      {
         delta[i8][1]=exz*nz;
         delta[i8][2]=eyz*nz;
         delta[i8][3]=ezz*nz;
      }
      if( i8==3||i8==4 )
      {
         delta[i8][1]=exy*ny;
         delta[i8][2]=eyy*ny;
         delta[i8][3]=eyz*ny;
      }
      if( i8==7||i8==8 )
      {
         delta[i8][1]=exy*ny+exz*nz;
         delta[i8][2]=eyy*ny+eyz*nz;
         delta[i8][3]=eyz*ny+ezz*nz;
      }
   }
}
for ( i = 1; i <= nx-1; i++ )
{
   m=nxy*( nz-1 )+nx*( ny-1 )+i;
   for (int nn = 1; nn <= 3; nn++ )
   {
      for (int mm = 1; mm <= 8; mm++ )
      {
         double sum=0.0;
         for ( m3 = 1; m3 <= 3; m3++ )
         {
            for (int m8 = 1; m8 <= 8; m8++ )
            {
               if (phasechange[m]==1)
               {
                  for (qtime=pixt[m]+1; qtime <=q; qtime++)
                  {
                     sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                        [nn][qtime-pixt[m]];
```

```
                    C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                       [nn][qtime-pixt[m]]*delta[mm][nn];
                  }
               }

               if (phasechange[m]==0)
               {
                  for (qtime=1; qtime <=q; qtime++)
                  {
                     sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime];

                     C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime]
                       *delta[mm][nn];
                  }
               }
            }
         }
         b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
      }
   }
}
/*
** x=nx y=ny z=nz corner
*/
for ( i3 = 1; i3 <= 3; i3++ )
{
   for (int i8 = 1; i8 <= 8; i8++ )
   {
      delta[i8][i3]=0.0;
      if( i8==2 )
      {
         delta[i8][1]=exx*nx;
         delta[i8][2]=exy*nx;
         delta[i8][3]=exz*nx;
      }
      if( i8==4 )
      {
         delta[i8][1]=exy*ny;
         delta[i8][2]=eyy*ny;
         delta[i8][3]=eyz*ny;
      }
      if( i8==5 )
      {
         delta[i8][1]=exz*nz;
```

```
            delta[i8][2]=eyz*nz;
            delta[i8][3]=ezz*nz;
         }
         if( i8==8 )
         {
            delta[i8][1]=exy*ny+exz*nz;
            delta[i8][2]=eyy*ny+eyz*nz;
            delta[i8][3]=eyz*ny+ezz*nz;
         }
         if( i8==6 )
         {
            delta[i8][1]=exx*nx+exz*nz;
            delta[i8][2]=exy*nx+eyz*nz;
            delta[i8][3]=exz*nx+ezz*nz;
         }
         if( i8==3 )
         {
            delta[i8][1]=exx*nx+exy*ny;
            delta[i8][2]=exy*nx+eyy*ny;
            delta[i8][3]=exz*nx+eyz*ny;
         }
         if( i8==7 )
         {
            delta[i8][1]=exx*nx+exy*ny+exz*nz;
            delta[i8][2]=exy*nx+eyy*ny+eyz*nz;
            delta[i8][3]=exz*nx+eyz*ny+ezz*nz;
         }
      }
   }
m=nx*ny*nz;
for (int nn = 1; nn <= 3; nn++ )
{
   for (int mm = 1; mm <= 8; mm++ )
   {
      double sum=0.0;
      for ( m3 = 1; m3 <= 3; m3++ )
      {
         for (int m8 = 1; m8 <= 8; m8++ )
         {
            if (phasechange[m]==1)
            {
               for (qtime=pixt[m]+1; qtime <=q; qtime++)
               {
                  sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm]
```

```
                        [nn][qtime-pixt[m]];

                    C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm]
                        [nn][qtime-pixt[m]]*delta[mm][nn];
                }
            }

            if (phasechange[m]==0)
            {
                for (qtime=1; qtime <=q; qtime++)
                {
                    sum=sum+delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime];

                    C=C+0.5*delta[m8][m3]*dk[pix[m]][m8][m3][mm][nn][qtime]
                      *delta[mm][nn];
                }
            }
        }

      }
      b[ib[m][is[mm]]][nn]=b[ib[m][is[mm]]][nn]+sum;
    }
  }
}

/*
** Subroutine computes the total energy, utot, and the gradient, gb.
*/
void energy(int nx, int ny, int nz, int ns, double &utot, int q )
{
  for ( int m3 = 1; m3 <= 3; m3++ )
  {
    for (int m = 1; m <= ns; m++ )
    {
      gb[m][m3]=0.0;
    }
  }

  /*
  ** Do global matrix multiply via small stiffness matrices,
  **a part of gb is A(t(1))*u(t(q)).
  ** The long statement below correctly brings in all the terms from the global
  ** matrix A using only the small stiffness matrices when only considering the
  ** deformation at time equals t(q).
```

```
*/

for ( int j = 1; j <= 3; j++ )
{
   for ( int n = 1; n <= 3; n++ )
   {
      for (int m = 1; m <= ns; m++ )
      {
         for (int qtime = 1; qtime <= 1; qtime++ )
         {
            gb[m][j]=gb[m][j]+u[ib[m][1]][q+1-qtime][n]*(
                        dk[pix[ib[m][27]]][1][j][4][n][qtime]+
                        dk[pix[ib[m][7]]][2][j][3][n][qtime]+
                        dk[pix[ib[m][25]]][5][j][8][n][qtime]+
                        dk[pix[ib[m][15]]][6][j][7][n][qtime])+
                        u[ib[m][2]][q+1-qtime][n]*(
                        dk[pix[ib[m][27]]][1][j][3][n][qtime]+
                        dk[pix[ib[m][25]]][5][j][7][n][qtime])+
                        u[ib[m][3]][q+1-qtime][n]*(
                        dk[pix[ib[m][27]]][1][j][2][n][qtime]+
                        dk[pix[ib[m][5]]][4][j][3][n][qtime]+
                        dk[pix[ib[m][13]]][8][j][7][n][qtime]+
                        dk[pix[ib[m][25]]][5][j][6][n][qtime])+
                        u[ib[m][4]][q+1-qtime][n]*(
                        dk[pix[ib[m][5]]][4][j][2][n][qtime]+
                        dk[pix[ib[m][13]]][8][j][6][n][qtime])+
                        u[ib[m][5]][q+1-qtime][n]*(
                        dk[pix[ib[m][6]]][3][j][2][n][qtime]+
                        dk[pix[ib[m][5]]][4][j][1][n][qtime]+
                        dk[pix[ib[m][14]]][7][j][6][n][qtime]+
                        dk[pix[ib[m][13]]][8][j][5][n][qtime])+
                        u[ib[m][6]][q+1-qtime][n]*(
                        dk[pix[ib[m][6]]][3][j][1][n][qtime] +
                        dk[pix[ib[m][14]]][7][j][5][n][qtime])+
                        u[ib[m][7]][q+1-qtime][n]*(
                        dk[pix[ib[m][6]]][3][j][4][n][qtime]+
                        dk[pix[ib[m][7]]][2][j][1][n][qtime]+
                        dk[pix[ib[m][14]]][7][j][8][n][qtime]+
                        dk[pix[ib[m][15]]][6][j][5][n][qtime])+
                        u[ib[m][8]][q+1-qtime][n]*(
                        dk[pix[ib[m][7]]][2][j][4][n][qtime]+
                        dk[pix[ib[m][15]]][6][j][8][n][qtime])+
                        u[ib[m][9]][q+1-qtime][n]*(
                        dk[pix[ib[m][25]]][5][j][4][n][qtime]+
```

```
dk[pix[ib[m][15]]][6][j][3][n][qtime])+
u[ib[m][10]][q+1-qtime][n]*(
dk[pix[ib[m][25]]][5][j][3][n][qtime])+
u[ib[m][11]][q+1-qtime][n]*(
dk[pix[ib[m][13]]][8][j][3][n][qtime]+
dk[pix[ib[m][25]]][5][j][2][n][qtime])+
u[ib[m][12]][q+1-qtime][n]*(
dk[pix[ib[m][13]]][8][j][2][n][qtime])+
u[ib[m][13]][q+1-qtime][n]*(
dk[pix[ib[m][13]]][8][j][1][n][qtime]+
dk[pix[ib[m][14]]][7][j][2][n][qtime])+
u[ib[m][14]][q+1-qtime][n]*(
dk[pix[ib[m][14]]][7][j][1][n][qtime])+
u[ib[m][15]][q+1-qtime][n]*(
dk[pix[ib[m][14]]][7][j][4][n][qtime]+
dk[pix[ib[m][15]]][6][j][1][n][qtime])+
u[ib[m][16]][q+1-qtime][n]*(
dk[pix[ib[m][15]]][6][j][4][n][qtime])+
u[ib[m][17]][q+1-qtime][n]*(
dk[pix[ib[m][27]]][1][j][8][n][qtime]+
dk[pix[ib[m][7]]][2][j][7][n][qtime])+
u[ib[m][18]][q+1-qtime][n]*(
dk[pix[ib[m][27]]][1][j][7][n][qtime])+
u[ib[m][19]][q+1-qtime][n]*(
dk[pix[ib[m][27]]][1][j][6][n][qtime]+
dk[pix[ib[m][5]]][4][j][7][n][qtime])+
u[ib[m][20]][q+1-qtime][n]*(
dk[pix[ib[m][5]]][4][j][6][n][qtime])+
u[ib[m][21]][q+1-qtime][n]*(
dk[pix[ib[m][5]]][4][j][5][n][qtime]+
dk[pix[ib[m][6]]][3][j][6][n][qtime])+
u[ib[m][22]][q+1-qtime][n]*(
dk[pix[ib[m][6]]][3][j][5][n][qtime])+
u[ib[m][23]][q+1-qtime][n]*(
dk[pix[ib[m][6]]][3][j][8][n][qtime]+
dk[pix[ib[m][7]]][2][j][5][n][qtime])+
u[ib[m][24]][q+1-qtime][n]*(
dk[pix[ib[m][7]]][2][j][8][n][qtime])+
u[ib[m][25]][q+1-qtime][n]*(
dk[pix[ib[m][14]]][7][j][3][n][qtime]+
dk[pix[ib[m][13]]][8][j][4][n][qtime]+
dk[pix[ib[m][15]]][6][j][2][n][qtime]+
dk[pix[ib[m][25]]][5][j][1][n][qtime])+
u[ib[m][26]][q+1-qtime][n]*(
```

```
                        dk[pix[ib[m][6]]][3][j][7][n][qtime]+
                        dk[pix[ib[m][5]]][4][j][8][n][qtime]+
                        dk[pix[ib[m][27]]][1][j][5][n][qtime]+
                        dk[pix[ib[m][7]]][2][j][6][n][qtime])+
                        u[ib[m][27]][q+1-qtime][n]*(
                        dk[pix[ib[m][27]]][1][j][1][n][qtime]+
                        dk[pix[ib[m][7]]][2][j][2][n][qtime]+
                        dk[pix[ib[m][6]]][3][j][3][n][qtime]+
                        dk[pix[ib[m][5]]][4][j][4][n][qtime]+
                        dk[pix[ib[m][25]]][5][j][5][n][qtime]+
                        dk[pix[ib[m][15]]][6][j][6][n][qtime]+
                        dk[pix[ib[m][14]]][7][j][7][n][qtime]+
                        dk[pix[ib[m][13]]][8][j][8][n][qtime]);
            }
        }
    }
}

/*
** Calculate for b and C.
*/
utot=C;

for ( m3 = 1; m3 <= 3; m3++ )
{
    for (int m = 1; m <= ns; m++ )
    {
        if(phasechange[m]==1)
        {
            utot=utot+0.5*(u[m][q][m3]-u[m][pixt[m]][m3])
                *gb[m][m3]+b[m][m3]*(u[m][q][m3]-u[m][pixt[m]][m3]);
        }

        if(phasechange[m]==0)
        {
            utot=utot+0.5*u[m][q][m3]
                *gb[m][m3]+b[m][m3]*u[m][q][m3];
        }
        gb[m][m3]=gb[m][m3]+b[m][m3];
    }
}
}

/*
```

```
** Subroutine that carries out the conjugate gradient relaxation process.
*/
void     dembx( int ns, int &Lstep, double &gg,double  (&dk)[101][9][4][9][4][51],
double &gtest, int &ldemb, int &kkk, int q )
{
   /*
   ** Initialize the conjugate direction vector on first call to dembx only
   ** For calls to dembx after the first, we want to continue using the
   ** value of h determined in the previous call. Of course, if npoints is
   ** greater than 1, this initialization step will be run for every new
   ** microstructure used, as kkk is reset to 1 every time the counter micro
   ** is increased.
   */
   double gamma;
   double lambda;

   int qtime;
   if( kkk==1 )
   {
      for (int  m3 = 1; m3 <= 3; m3++ )
      {
         for (int  m = 1; m <= ns; m++ )
         {
            h[m][m3]=gb[m][m3];
         }
      }
   }
   /*
   ** Lstep counts the number of conjugate gradient steps taken in
   ** each call to dembx.
   */
   Lstep=0;

   for (int  ijk = 1; ijk <= ldemb; ijk++ )
   {
      Lstep=Lstep+1;

      for (int m3 = 1; m3 <= 3; m3++ )
      {
         for (int m = 1; m <= ns; m++ )
         {
            Ah[m][m3]=0.0;
         }
      }
```

```
/*
** Do global matrix multiply via small stiffness matrices, Ah = A * h
** The long statement below correctly brings in all the terms from
** the global matrix A using only the small stiffness matrices dk.
*/
for (int j = 1; j <= 3; j++ )
{
   for (int  n = 1; n <= 3; n++ )
   {
      for (int m = 1; m <= ns; m++ )
      {
         qtime=1;

         Ah[m][j]=Ah[m][j]+h[ib[m][1]][n]*(dk[pix[ib[m][27]]][1][j][4][n][qtime] +
                  dk[pix[ib[m][7]]][2][j][3][n][qtime] +
                  dk[pix[ib[m][25]]][5][j][8][n][qtime]+
                  dk[pix[ib[m][15]]][6][j][7][n][qtime])+
                  h[ib[m][2]][n]*(dk[pix[ib[m][27]]][1][j][3][n][qtime] +
                  dk[pix[ib[m][25]]][5][j][7][n][qtime])+
                  h[ib[m][3]][n]*(dk[pix[ib[m][27]]][1][j][2][n][qtime]+
                  dk[pix[ib[m][5]]][4][j][3][n][qtime]+
                  dk[pix[ib[m][13]]][8][j][7][n][qtime]+
                  dk[pix[ib[m][25]]][5][j][6][n][qtime])+
                  h[ib[m][4]][n]*(dk[pix[ib[m][5]]][4][j][2][n][qtime] +
                  dk[pix[ib[m][13]]][8][j][6][n][qtime])+
                  h[ib[m][5]][n]*(dk[pix[ib[m][6]]][3][j][2][n][qtime]+
                  dk[pix[ib[m][5]]][4][j][1][n][qtime]+
                  dk[pix[ib[m][14]]][7][j][6][n][qtime]+
                  dk[pix[ib[m][13]]][8][j][5][n][qtime])+
                  h[ib[m][6]][n]*(dk[pix[ib[m][6]]][3][j][1][n][qtime] +
                  dk[pix[ib[m][14]]][7][j][5][n][qtime])+
                  h[ib[m][7]][n]*(dk[pix[ib[m][6]]][3][j][4][n][qtime]+
                  dk[pix[ib[m][7]]][2][j][1][n][qtime]+
                  dk[pix[ib[m][14]]][7][j][8][n][qtime]+
                  dk[pix[ib[m][15]]][6][j][5][n][qtime])+
                  h[ib[m][8]][n]*(dk[pix[ib[m][7]]][2][j][4][n][qtime] +
                  dk[pix[ib[m][15]]][6][j][8][n][qtime])+
                  h[ib[m][9]][n]*(dk[pix[ib[m][25]]][5][j][4][n][qtime] +
                  dk[pix[ib[m][15]]][6][j][3][n][qtime])+
                  h[ib[m][10]][n]*(dk[pix[ib[m][25]]][5][j][3][n][qtime])+
                  h[ib[m][11]][n]*(dk[pix[ib[m][13]]][8][j][3][n][qtime] +
                  dk[pix[ib[m][25]]][5][j][2][n][qtime])+
                  h[ib[m][12]][n]*(dk[pix[ib[m][13]]][8][j][2][n][qtime])+
                  h[ib[m][13]][n]*(dk[pix[ib[m][13]]][8][j][1][n][qtime] +
```

```
                    dk[pix[ib[m][14]]][7][j][2][n][qtime])+
                    h[ib[m][14]][n]*(dk[pix[ib[m][14]]][7][j][1][n][qtime])+
                    h[ib[m][15]][n]*(dk[pix[ib[m][14]]][7][j][4][n][qtime] +
                    dk[pix[ib[m][15]]][6][j][1][n][qtime])+
                    h[ib[m][16]][n]*(dk[pix[ib[m][15]]][6][j][4][n][qtime])+
                    h[ib[m][17]][n]*(dk[pix[ib[m][27]]][1][j][8][n][qtime] +
                    dk[pix[ib[m][7]]][2][j][7][n][qtime])+
                    h[ib[m][18]][n]*(dk[pix[ib[m][27]]][1][j][7][n][qtime])+
                    h[ib[m][19]][n]*(dk[pix[ib[m][27]]][1][j][6][n][qtime] +
                    dk[pix[ib[m][5]]][4][j][7][n][qtime])+
                    h[ib[m][20]][n]*(dk[pix[ib[m][5]]][4][j][6][n][qtime])+
                    h[ib[m][21]][n]*(dk[pix[ib[m][5]]][4][j][5][n][qtime] +
                    dk[pix[ib[m][6]]][3][j][6][n][qtime])+
                    h[ib[m][22]][n]*(dk[pix[ib[m][6]]][3][j][5][n][qtime])+
                    h[ib[m][23]][n]*(dk[pix[ib[m][6]]][3][j][8][n][qtime] +
                    dk[pix[ib[m][7]]][2][j][5][n][qtime])+
                    h[ib[m][24]][n]*(dk[pix[ib[m][7]]][2][j][8][n][qtime])+
                    h[ib[m][25]][n]*(dk[pix[ib[m][14]]][7][j][3][n][qtime] +
                    dk[pix[ib[m][13]]][8][j][4][n][qtime]+
                    dk[pix[ib[m][15]]][6][j][2][n][qtime]+
                    dk[pix[ib[m][25]]][5][j][1][n][qtime])+
                    h[ib[m][26]][n]*(dk[pix[ib[m][6]]][3][j][7][n][qtime] +
                    dk[pix[ib[m][5]]][4][j][8][n][qtime]+
                    dk[pix[ib[m][27]]][1][j][5][n][qtime]+
                    dk[pix[ib[m][7]]][2][j][6][n][qtime])+
                    h[ib[m][27]][n]*(dk[pix[ib[m][27]]][1][j][1][n][qtime] +
                    dk[pix[ib[m][7]]][2][j][2][n][qtime]+
                    dk[pix[ib[m][6]]][3][j][3][n][qtime]+
                    dk[pix[ib[m][5]]][4][j][4][n][qtime] +
                    dk[pix[ib[m][25]]][5][j][5][n][qtime]+
                    dk[pix[ib[m][15]]][6][j][6][n][qtime]+
                    dk[pix[ib[m][14]]][7][j][7][n][qtime]+
                    dk[pix[ib[m][13]]][8][j][8][n][qtime]);
      }
   }
}

double hAh=0.0;
for ( m3 = 1; m3 <= 3; m3++ )
{
   for (int m = 1; m <= ns; m++ )
   {
      hAh=hAh+h[m][m3] *Ah[m][m3];
   }
```

```
      }

      lambda=gg/hAh;
      for ( m3 = 1; m3 <= 3; m3++ )
      {
         for (int m = 1; m <= ns; m++ )
         {
            u[m][q][m3]=u[m][q][m3]-lambda*h[m][m3];

            gb[m][m3]=gb[m][m3]-lambda*Ah[m][m3];
         }
      }

      double gglast=gg;
      gg=0.0;
      for ( m3 = 1; m3 <= 3; m3++ )
      {
         for (int m = 1; m <= ns; m++ )
         {
            gg=gg+gb[m][m3]*gb[m][m3];
         }
      }
      if( gg<gtest )
      {
         break;
      }

      gamma=gg/gglast;
      for ( m3 = 1; m3 <= 3; m3++ )
      {
         for (int m = 1; m <= ns; m++ )
         {
            h[m][m3]=gb[m][m3]+gamma*h[m][m3];
         }
      }
   }
}

/*
** Subroutine that computes the six average stresses and six
** average strains.
*/
void stress(int nx,int ny,int  nz,int ns,int q )
{
```

```
double  dndx[9],dndy[9],dndz[9],es[7][9][4],uu[9][51][4];

int nxy=nx*ny;

/*
** Set up single element strain matrix.
** dndx, dndy, and dndz are the components of the average strain
** matrix in a pixel.
*/

dndx[1]=-0.25;
dndx[2]=0.25;
dndx[3]=0.25;
dndx[4]=-0.25;
dndx[5]=-0.25;
dndx[6]=0.25;
dndx[7]=0.25;
dndx[8]=-0.25;
dndy[1]=-0.25;
dndy[2]=-0.25;
dndy[3]=0.25;
dndy[4]=0.25;
dndy[5]=-0.25;
dndy[6]=-0.25;
dndy[7]=0.25;
dndy[8]=0.25;
dndz[1]=-0.25;
dndz[2]=-0.25;
dndz[3]=-0.25;
dndz[4]=-0.25;
dndz[5]=0.25;
dndz[6]=0.25;
dndz[7]=0.25;
dndz[8]=0.25;
/*
** Build averaged strain matrix, follows code in femat, but for average
** strain over the pixel, not the strain at a point.
*/
for (int n1 = 1; n1 <= 6; n1++ )
{
   for (int n2 = 1; n2 <= 8; n2++ )
   {
      for (int n3 = 1; n3 <= 3; n3++ )
      {
```

```
            es[ n1][ n2][n3]=0.0;

         }
      }
   }
for (int n = 1; n <= 8; n++ )
{
   es[1][n][1]=dndx[n];
   es[2][n][2]=dndy[n];
   es[3][n][3]=dndz[n];
   es[4][n][1]=dndz[n];
   es[4][n][3]=dndx[n];
   es[5][n][2]=dndz[n];
   es[5][n][3]=dndy[n];
   es[6][n][1]=dndy[n];
   es[6][n][2]=dndx[n];
}

/*
** Compute components of the average stress and strain tensors in each pixel.
*/
strxx=0.0;
stryy=0.0;
strzz=0.0;
strxz=0.0;
stryz=0.0;
strxy=0.0;
sxx=0.0;
syy=0.0;
szz=0.0;
sxz=0.0;
syz=0.0;
sxy=0.0;

int set;

for (int k = 1; k <= nz; k++ )
{
   for (int j = 1; j <= ny; j++ )
   {
      for (int i = 1; i <= nx; i++ )
      {
         int m=( k-1 )*nxy+( j-1 )*nx+i;
         str11=0.0;
```

```
str22=0.0;
str33=0.0;
str13=0.0;
str23=0.0;
str12=0.0;

if (phasechange[m]==1)
{
   if(pix[m]==7)set=pixt[m]+1;
   else set=q;

   for (int qtime=set; qtime <= q; qtime++ )
   {

      /*
      ** Load in elements of 8-vector using pd. bd. conds.
      */
      for (int mm =1; mm <= 3; mm++ )
      {
         uu[1][qtime][mm]=u[m][qtime][mm]-u[m][pixt[m]][mm];
         uu[2][qtime][mm]
         =u[ib[m][3]][qtime][mm]-u[ib[m][3]][pixt[m]][mm];
         uu[3][qtime][mm]
         =u[ib[m][2]][qtime][mm]-u[ib[m][2]][pixt[m]][mm];
         uu[4][qtime][mm]
         =u[ib[m][1]][qtime][mm]-u[ib[m][1]][pixt[m]][mm];
         uu[5][qtime][mm]
         =u[ib[m][26]][qtime][mm]-u[ib[m][26]][pixt[m]][mm];
         uu[6][qtime][mm]
         =u[ib[m][19]][qtime][mm]-u[ib[m][19]][pixt[m]][mm];
         uu[7][qtime][mm]
         =u[ib[m][18]][qtime][mm]-u[ib[m][18]][pixt[m]][mm];
         uu[8][qtime][mm]
         =u[ib[m][17]][qtime][mm]-u[ib[m][17]][pixt[m]][mm];
      }

      /*
      ** Local stresses and strains in a pixel.
      */
      for (int n3 =1; n3 <= 3; n3++ )
      {
         for (int n8 =1; n8 <= 8; n8++ )
         {
            for ( n =1; n <= 6; n++ )
```

```
            {
               str11=str11+cmod[pix[m]][1][n][q+1-qtime]
               *es[n][n8][n3]*uu[n8][qtime][n3];
               str22=str22+cmod[pix[m]][2][n][q+1-qtime]
                    *es[n][n8][n3]*uu[n8][qtime][n3];
               str33=str33+cmod[pix[m]][3][n][q+1-qtime]
                    *es[n][n8][n3]*uu[n8][qtime][n3];
               str13=str13+cmod[pix[m]][4][n][q+1-qtime]
                    *es[n][n8][n3]*uu[n8][qtime][n3];
               str23=str23+cmod[pix[m]][5][n][q+1-qtime]
                    *es[n][n8][n3]*uu[n8][qtime][n3];
               str12=str12+cmod[pix[m]][6][n][q+1-qtime]
                    *es[n][n8][n3]*uu[n8][qtime][n3];
            }
         }
      }
   }
}

if (phasechange[m]==0)
{
   if(pix[m]==7)set=1;
   else set=q;

   for (int qtime=set; qtime <= q; qtime++ )
   {

      /*
      ** load in elements of 8-vector using pd. bd. conds.
      */
      for (int mm =1; mm <= 3; mm++ )
      {

         uu[1][qtime][mm]=u[m][qtime][mm];
         uu[2][qtime][mm]
         =u[ib[m][3]][qtime][mm];
         uu[3][qtime][mm]
         =u[ib[m][2]][qtime][mm];
         uu[4][qtime][mm]
         =u[ib[m][1]][qtime][mm];
         uu[5][qtime][mm]
         =u[ib[m][ 26 ]][qtime][mm];
         uu[6][qtime][mm]
         =u[ib[m][ 19 ]][qtime][mm];
```

```
   uu[7][qtime][mm]
   =u[ib[m][ 18 ]][qtime][mm];
   uu[8][qtime][mm]
   =u[ib[m][17]][qtime][mm];
}

/*
**Correction for period boundary condition.
*/
if( i==nx )
{
   uu[2][qtime][1]
   =uu[2][qtime][1]+exx*nx;
   uu[2][qtime][2]
   =uu[2][qtime][2]+exy*nx;
   uu[2][qtime][3]
   =uu[2][qtime][3]+exz*nx;
   uu[3][qtime][1]
   =uu[3][qtime][1]+exx*nx;
   uu[3][qtime][2]
   =uu[3][qtime][2]+exy*nx;
   uu[3][qtime][3]
   =uu[3][qtime][3]+exz*nx;
   uu[6][qtime][1]
   =uu[6][qtime][1]+exx*nx;
   uu[6][qtime][2]
   =uu[6][qtime][2]+exy*nx;
   uu[6][qtime][3]
   =uu[6][qtime][3]+exz*nx;
   uu[7][qtime][1]
   =uu[7][qtime][1]+exx*nx;
   uu[7][qtime][2]
   =uu[7][qtime][2]+exy*nx;
   uu[7][qtime][3]
   =uu[7][qtime][3]+exz*nx;

}
if( j==ny )
{
   uu[3][qtime][1]
   =uu[3][qtime][1]+exy*ny;
   uu[3][qtime][2]
   =uu[3][qtime][2]+eyy*ny;
   uu[3][qtime][3]
```

```
   =uu[3][qtime][3]+eyz*ny;
   uu[4][qtime][1]
   =uu[4][qtime][1]+exy*ny;
   uu[4][qtime][2]
   =uu[4][qtime][2]+eyy*ny;
   uu[4][qtime][3]
   =uu[4][qtime][3]+eyz*ny;
   uu[7][qtime][1]
   =uu[7][qtime][1]+exy*ny;
   uu[7][qtime][2]
   =uu[7][qtime][2]+eyy*ny;
   uu[7][qtime][3]
   =uu[7][qtime][3]+eyz*ny;
   uu[8][qtime][1]
   =uu[8][qtime][1]+exy*ny;
   uu[8][qtime][2]
   =uu[8][qtime][2]+eyy*ny;
   uu[8][qtime][3]
   =uu[8][qtime][3]+eyz*ny;

}
if( k==nz )
{
   uu[5][qtime][1]
   =uu[5][qtime][1]+exz*nz;
   uu[5][qtime][2]
   =uu[5][qtime][2]+eyz*nz;
   uu[5][qtime][3]
   =uu[5][qtime][3]+ezz*nz;
   uu[6][qtime][1]
   =uu[6][qtime][1]+exz*nz;
   uu[6][qtime][2]
   =uu[6][qtime][2]+eyz*nz;
   uu[6][qtime][3]
   =uu[6][qtime][3]+ezz*nz;
   uu[7][qtime][1]
   =uu[7][qtime][1]+exz*nz;
   uu[7][qtime][2]
   =uu[7][qtime][2]+eyz*nz;
   uu[7][qtime][3]
   =uu[7][qtime][3]+ezz*nz;
   uu[8][qtime][1]
   =uu[8][qtime][1]+exz*nz;
   uu[8][qtime][2]
```

```
            =uu[8][qtime][2]+eyz*nz;
            uu[8][qtime][3]
            =uu[8][qtime][3]+ezz*nz;
         }

         /*
          ** Local stresses and strains in a pixel.
         */
         for (int n3 =1; n3 <= 3; n3++ )
         {
            for (int n8 =1; n8 <= 8; n8++ )
            {
               for ( n =1; n <= 6; n++ )
               {
                  str11=str11+cmod[pix[m]
                        ][1][n][q+1-qtime]
                        *es[n][n8][n3]
                        *uu[n8][qtime][n3];
                  str22=str22+cmod[pix[m]
                        ][2][n][q+1-qtime]
                        *es[n][n8][n3]*uu[n8][qtime][n3];
                  str33=str33+cmod[pix[m]
                        ][3][n][q+1-qtime]
                        *es[n][n8][n3]*uu[n8][qtime][n3];
                  str13=str13+cmod[pix[m]
                        ][4][n][q+1-qtime]
                        *es[n][n8][n3]*uu[n8][qtime][n3];
                  str23=str23+cmod[pix[m]
                        ][5][n][q+1-qtime]
                        *es[n][n8][n3]*uu[n8][qtime][n3];
                  str12=str12+cmod[pix[m]
                        ][6][n][q+1-qtime]
                        *es[n][n8][n3]*uu[n8][qtime][n3];
               }
            }
         }
      }
   }

   /*
   ** Sum up local strains and stresses into global values.
   */
   strxx=strxx+str11;
   stryy=stryy+str22;
```

```
            strzz=strzz+str33;
            strxz=strxz+str13;
            stryz=stryz+str23;
            strxy=strxy+str12;
         }
      }
   }

   /*
   ** Volume average of global stresses and strains.
   */
   strxx=strxx/(1.0* ns) ;
   stryy=stryy/(1.0* ns);
   strzz=strzz/(1.0* ns);
   strxz=strxz/(1.0* ns);
   stryz=stryz/(1.0* ns);
   strxy=strxy/(1.0* ns);
   sxx=exx;
   syy=eyy;
   szz=ezz;
   sxz=exz;
   syz=eyz;
   sxy=exy;
}

/*
**(USER)Subfuction for inputting modulus function for viscoelastic materials.
*/
double Fmodulus(double t)
{
   return 11.2+11.2*exp(-0.2*t);
}

/*
** The main function
*/
void main()
{
   /*
   **Local variables for the main function.
   */
   double phasemod[101][51][3],prob[101];
   int in[28];
   int jn[28];
```

```
int kn[28];

int nphase=8;//(USER)Number of phases in the composite material

/*
** (USER)
** Set time steps. When aging is not considered, for viscoelastic materials,
** Log time is better used than normal time. When aging occures, a constant time
** change rate is required.
**
*/
int step=8;
double t[101];
double tmin=0.05;
double tmax=50;
for (int q=1; q<=step; q++)
{
   t[q]=tmin*pow(tmax/tmin,(q*1.0-1.)/(step*1.0-1.));
}

/*
**(User)Read in files for different microstructure. The name of files can be changed.
*/
string prefilename[50];
char filename[15];

prefilename[1]="1.txt";
prefilename[2]="2.txt";
prefilename[3]="3.txt";
prefilename[4]="4.txt";
prefilename[5]="5.txt";
prefilename[6]="6.txt";
prefilename[7]="7.txt";
prefilename[8]="8.txt";
prefilename[9]="9.txt";
prefilename[10]="10.txt";
prefilename[11]="11.txt";
prefilename[12]="12.txt";
prefilename[13]="13.txt";
prefilename[14]="14.txt";
prefilename[15]="15.txt";
prefilename[16]="16.txt";
prefilename[17]="17.txt";
prefilename[18]="18.txt";
```

```
prefilename[19]="19.txt";
prefilename[20]="20.txt";
prefilename[21]="21.txt";
prefilename[22]="22.txt";
prefilename[23]="23.txt";
prefilename[24]="24.txt";

/*
**(USER)Composite's 3D dimension.
*/
int nx=100;
int ny=100;
int nz=100;

/*
**ns = Total number of sites.
*/
int ns=nx*ny*nz;

/*
**Assign to each pixel that aging has not occured by setting the phaechange value to
**be 0 and the time for aging occures is 0.
*/
int nxy=nx*ny;
for(int k=1; k<=nz; k++)
{
   for (int j=1; j<=ny; j++)
   {
      for (int i=1; i<=nx; i++)
      {
         int m=nxy*(k-1)+nx*(j-1)+i;
         phasechange[m]=0;
         pixt[m]=0;
      }
   }
}

/*
**
** (USER)
** The parameter phasemod[i][j][k] is the Bulk [i][j][1] and Shear [i][j][2] moduli of
** the i'th phase at time step = j. These can be input in terms of Young's moduli
** E[i][j][1] andPoisson's ratio nu [i][j][2].  The program then changes them
** to Bulk and Shear moduli, using relations for isotropic elasticmoduli.
```

```
** For anisotropic elastic material, one can directly inputthe elastic moduli tensor
** cmod in subroutine femat, and skip this part.
**
*/
for ( int j = 1; j <= step; j++ )
{
   for ( int i = 1; i <= nphase; i++ )
   {
      phasemod[i][j][1]=0.;
      phasemod[i][j][2]=0.;
   }
}

/*
**When Poisson's ratio is not constant,laplace transform should be used to get the
** correct Bulk and Shear modulus.
*/
phasemod[1][1][1]=0.0;
phasemod[1][1][2]=0.5;
phasemod[2][1][1]=117.6;
phasemod[2][1][2]=0.314;
phasemod[3][1][1]=42.3;
phasemod[3][1][2]=0.324;
phasemod[4][1][1]=45.7;
phasemod[4][1][2]=0.33;
phasemod[5][1][1]=62.85;
phasemod[5][1][2]=0.3;
phasemod[6][1][1]=80.0;
phasemod[6][1][2]=0.275;
phasemod[7][1][1]=Fmodulus(t[1]);
phasemod[7][1][2]=0.25;
phasemod[8][1][1]=22.4;
phasemod[8][1][2]=0.25;

/*
** Modulus for viscoelastic pixels are the differences between time steps at time step
** greater than one. Poisson's ratios are kept constant.
*/
for( int i=2; i <= step; i++)
{
   for ( int j = 1; j <= nphase; j++ )
   {
      phasemod[7][i][1]=Fmodulus(t[i])-Fmodulus(t[i-1]);
      phasemod[j][i][2]=phasemod[j][1][2];
```

```
    }
 }

 /*
 ** If inputs are Yong's modulus and Poission's ration, use this loop. If inputs are
 ** Bulk and Shear modulus, skip this loop.
 */
for ( i = 1; i <= nphase; i++ )
 {
    for ( j = 1; j <= step; j++ )
    {
       double save=phasemod[i][j][1];
       phasemod[i][j][1]=phasemod[i][j][1]/3.0/( 1.0-2.0*phasemod[i][j][2] );
       phasemod[i][j][2]=save/2.0/( 1.0+phasemod[i][j][2] );
    }
 }

 /*
 ** (USER) Set applied strains
 ** Actual shear strain applied in loop is exy, exz, and eyz as
 ** given in the statements below.  The engineering shear strain, by which
 ** the Shear modulus is usually defined, is twice these values.
 */
 exx=0.1;
 eyy=0.1;
 ezz=0.1;
 exz=0.05;
 eyz=0.05;
 exy=0.05;

 ofstream fout("outputfile.txt");

 /*
 **TIME LOOP Starts!!!
 */
 for ( q = 1; q <= step; q++ )
 {
    strcpy (filename, prefilename[q].c_str());

    ifstream fin(filename);

    /*
    ** (USER) gtest is the stopping criterion, the number
    ** to which the quantity gg=gb*gb is compared.
```

```
** Usually gtest = abc*ns, so that when gg < gtest, the rms value
** per pixel of gb is less than sqrt(abc).
*/
double gtest=pow(10,-12)*ns;

/*
** Read in a microstructure in subroutine ppixel, and set up pix(m)
** with the appropriate phase assignments.
*/
ppixel( nx, ny, nz, ns, nphase,filename );

assig( ns, nphase, prob );

nxy=nx*ny;

/*
** Check whether aging occurs and store the time for aging occurs in pixt.
*/
for(int k=1; k<=nz; k++)
{
   for (int j=1; j<=ny; j++)
   {
      for (int i=1; i<=nx; i++)
      {
         int m=nxy*(k-1)+nx*(j-1)+i;
         if(q>1)
         {
            if(pix[m]!=pixstor[m])
            {
               phasechange[m]=1;
               pixt[m]=q-1;
            }
         }
         pixstor[m]=pix[m];
      }
   }
}

/*
** Construct the neighbor table, ib(m,n).
*/

/*
** First construct the 27 neighbor table in terms of delta i, delta j, and
```

```
** delta k information.
*/

in[1]=0;
in[2]=1;
in[3]=1;
in[4]=1;
in[5]=0;
in[6]=-1;
in[7]=-1;
in[8]=-1;

jn[1]=1;
jn[2]=1;
jn[3]=0;
jn[4]=-1;
jn[5]=-1;
jn[6]=-1;
jn[7]=0;
jn[8]=1;

for (int n = 1; n <= 8; n++ )
{
   kn[n]=0;
   kn[ n+8 ]=-1;
   kn[ n+16 ]=1;
   in[ n+8 ]=in[n];
   in[ n+16 ]=in[n];
   jn[ n+8 ]=jn[n ];
   jn[ n+16 ]=jn[n];
}

in[25]=0;
in[ 26 ]=0;
in[27]=0;
jn[25]=0;
jn[ 26 ]=0;
jn[27]=0;
kn[25]=-1;
kn[ 26 ]=1;
kn[27]=0;

/*
** Now construct neighbor table according to 1-d labels.
```

```
** Matrix ib(m,n) gives the 1-d label of the n'th neighbor (n=1,27) of
** the node labelled m.
*/

for ( k = 1; k <= nz; k++ )
{
   for (int j = 1; j <= ny; j++ )
   {
      for ( i = 1; i <= nx; i++ )
      {
         int m=nxy*( k-1 )+nx*( j-1 )+i;
         for ( n = 1; n <= 27; n++ )
         {
            int i1=i+in[n];
            int j1=j+jn[n];
            int k1=k+kn[n];
            if( i1<1 ) i1=i1+nx;
            if( i1>nx ) i1=i1-nx;
            if( j1<1 ) j1=j1+ny;
            if( j1>ny ) j1=j1-ny;
            if( k1<1 ) k1=k1+nz;
            if( k1>nz ) k1=k1-nz;
            int m1=nxy*( k1-1 )+nx*( j1-1 )+i1;
            ib[m][n]=m1;
         }
      }
   }
}

/*
**Compute the average stress and strain in each microstructure.
*/

/*
** Set up the modulus variables, finite element stiffness matrices,
** the constant, C, and vector, b, required for computing the energy.
** (USER) If anisotropic elastic moduli tensors are used, these need to be
** input in subroutine femat.
*/
femat( nx, ny, nz, ns, phasemod, nphase,q );
/*
** Apply chosen strains as a homogeneous macroscopic strain
** as the initial condition.
*/
```

```
for ( k = 1; k <= nz; k++ )
{
   for (int j = 1; j <= ny; j++ )
   {
      for ( i = 1; i <= nx; i++ )
      {
         int nxy=nx*ny;
         int m=nxy*( k-1 )+nx*( j-1 )+i;
         double x=1.0*( i-1 );
         double y=1.0*( j-1 );
         double z=1.0*( k-1 );
         u[m][q][1]=x*exx+y*exy+z*exz;
         u[m][q][2]=x*exy+y*eyy+z*eyz;
         u[m][q][3]=x*exz+y*eyz+z*ezz;
      }
   }
}

/*
** RELAXATION LOOP
** (USER) kmax is the maximum number of times dembx will be called, with
** ldemb conjugate gradient steps performed during each call.  The total
** number of conjugate gradient steps allowed for a given elastic
** computation is kmax*ldemb.
*/
int kmax=15,
    ldemb=50,
    ltot=0;
double utot;
int Lstep;
/*
** Call energy to get initial energy and initial gradient.
*/
energy( nx, ny, nz, ns, utot, q );
/*
** gg is the norm squared of the gradient (gg=gb*gb).
*/
double gg=0.0;

for ( int m3 = 1; m3 <= 3; m3++ )
{
   for (int m = 1; m <= ns; m++ )
   {
```

```
            gg=gg+gb[m][m3]*gb[m][m3];
        }
    }

    for (int kkk = 1; kkk <= kmax; kkk++ )
    {
        /*
        ** Call dembx to go into the conjugate gradient solver.
        */
        dembx( ns, Lstep, gg, dk, gtest, ldemb, kkk, q );
        ltot=ltot+Lstep;
        /*
        ** Call energy to compute energy after dembx call. If gg < gtest, this
        ** will be the final energy.  If gg is still larger than gtest, then this
        ** will give an intermediate energy with which to check how the
        ** relaxation process is coming along.
        */
        energy( nx, ny, nz, ns, utot, q );
        /*
        ** If relaxation process is finished, jump out of loop.
        */
        if( gg<=gtest ) break;
        /*
        ** If relaxation process will continue, compute and output stresses
        ** and strains as an additional aid to judge how the
        ** relaxation procedure is progressing.
        */
        stress( nx, ny, nz, ns,q );
    }
    stress( nx, ny, nz, ns,q );

    /*
    **Output calculated stresses.
    */
    fout<<"strxx      "<<" stryy       "<<"strzz       ";
    fout<<" strxz     "<<"stryz      "<<"strxy";
    fout<<"\n";

    fout<<strxx<<"    "<< stryy<<"    "<< strzz<<"    ";
    fout<< strxz<<"    "<< stryz<<"    "<< strxy;
    fout<<"\n";
  }
}
```

VITA

Xiaodan Li received her Bachelor of Civil Engineering from The Hong Kong University of Science and Technology in 2009. She entered the Civil Engineering Department at Texas A&M University in September 2009. Her research interests include modeling of aging viscoelastic properties of composite materials using finite element method.

Ms. Li may be reached at TTI 601R, Zachry Department of Civil Engineering, Texas A&M University, 3136 TAMU, College Station, Texas 77843-3136, USA. Her email is lxd19881102@gmail.com.